



Estd :1984

Department of INFORMATION TECHNOLOGY

22ITL41 Database Management Systems Laboratory Record

Name \_\_\_\_\_ Programme \_\_\_\_\_

Branch \_\_\_\_\_ Section \_\_\_\_\_ Semester \_\_\_\_\_

Roll No \_\_\_\_\_

*Certify that this is bonafide record of work done by the above student of*

*the 22ITL41- DATABASE MANAGEMENT SYSTEMS LABORATORY*

*during the year 2023 - 2024.*

Submitted for the Examination held on \_\_\_\_\_

Signature of the  
Lab Incharge

Head of the Department

Examiner 1

Examiner 2

# INDEX

EX. NO.	DATE	Name of Experiment	Marks (60)	Signature
1.	29.02.2024	DATA DEFINITION LANGUAGE COMMANDS, INTEGRITY CONSTRAINTS		
2.	07.03.2024	DATA MANIPULATION LANGUAGE, DATA CONTROL LANGUAGE COMMANDS AND TCL COMMANDS		
3.	28.03.2024	JOIN OPERATIONS		
4.	18.04.2024	NESTED QUERIES		
5.	18.04.2024	VIEWS AND INDEX		
6.	26.04.2024	STRING FUNCTIONS (Additional)		
7.	26.04.2024	AGGREGATE ORDER_BY, GROUP_BY AND HAVING CLAUSES AND SET OPERATIONS(Additional)		
8.	02.05.2024	DATE AND TIME FUNCTIONS(Additional)		
9.	09.05.2024	PL/SQL PROGRAMS		
10.	30.05.2024	CURSORS		
11.	06.06.2024	TRIGGERS		
12.	06.06.2024	PROCEDURES AND FUNCTIONS		
13.	13.06.2024	MINI PROJECT		

**EX.NO:01**

## **Data Definition Language commands and Integrity Constraints**

**29.02.2024**

### **AIM**

To execute Data Definition Language commands and Integrity Constraints.

### **INTEGRITY CONSTRAINT:**

#### **Create the Relation(table):**

```
SQL> create table student(roll number(10) primary key,name varchar(20) not null,mail  
varchar(20) unique,age number(3),gender char(1) check(gender='m' or gender='f'));
```

Table created.

```
SQL> insert into student values(&roll,&name,&mail,&age,&gender);
```

Enter value for roll: 1

Enter value for name: Dinesh

Enter value for mail: dinesh@gmail.com

Enter value for age: 18

Enter value for gender: m

```
old 1: insert into student values(&roll,&name,&mail,&age,&gender)
```

```
new 1: insert into student values(1,'Dinesh','dinesh@gmail.com',18,'m')
```

1 row created.

### **PRIMARY KEY CONSTRAINT**

```
SQL> /
```

Enter value for roll: 1

Enter value for name: Gunal

Enter value for mail: gunal@gmail.com

Enter value for age: 18

Enter value for gender: m

```
old 1: insert into student values(&roll,&name,&mail,&age,&gender)
```

```
new 1: insert into student values(1,'Gunal','gunal@gmail.com',18,'m')
```

```
insert into student values(1,'Gunal','gunal@gmail.com',18,'m')
```

\*

**ERROR at line 1:**

**ORA-00001: unique constraint (DINESHKUMAR.SYS\_C004061) violated**

SQL> /

Enter value for roll: 2

Enter value for name: Gunal

Enter value for mail: gunal@gmail.com

Enter value for age: 18

Enter value for gender: m

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(2,'Gunal','gunal@gmail.com',18,'m')

1 row created.

### **NOT NULL CONSTRAINT**

SQL> /

Enter value for roll: 3

Enter value for name:

Enter value for mail: babu@gmail.com

Enter value for age: 18

Enter value for gender: m

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(3,"','babu@gmail.com',18,'m')

insert into student values(3,"','babu@gmail.com',18,'m')

**ERROR at line 1:**

**ORA-01400: cannot insert NULL into ("DINESHKUMAR"."STUDENT"."NAME")**

SQL> /

Enter value for roll: 3

Enter value for name: babu

Enter value for mail: babu@gmail.com

Enter value for age: 18

Enter value for gender: m

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(3,'babu','babu@gmail.com',18,'m')

1 row created.

## **UNIQUE CONSTRAINT**

SQL> /

Enter value for roll: 4

Enter value for name: aanad

Enter value for mail: babu@gmail.com

Enter value for age: 18

Enter value for gender: m

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(4,'aanad','babu@gmail.com',18,'m')

insert into student values(4,'aanad','babu@gmail.com',18,'m')

\*

**ERROR at line 1:**

**ORA-00001: unique constraint (DINESHKUMAR.SYS\_C004062) violated**

Enter value for roll: 4

Enter value for name: aanad

Enter value for mail: aanad@gmail.com

Enter value for age: 18

Enter value for gender: m

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(4,'aanad','aanad@gmail.com',18,'m')

1 row created.

## **CHECK CONSTRAINT**

SQL> insert into student values(&roll,&name,&mail,&age,&gender');

Enter value for roll: 5

Enter value for name: suji

Enter value for mail: suji@gmail.com

Enter value for age: 18

Enter value for gender: F

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(5,'suji','suji@gmail.com',18,'F')

insert into student values(5,'suji','suji@gmail.com',18,'F')

\*

**ERROR at line 1:**

**ORA-02290: check constraint (DINESHKUMAR.SYS\_C004060) violated**

SQL> /

Enter value for roll: 5

Enter value for name: suji

Enter value for mail: suji@gmail.com

Enter value for age: 18

Enter value for gender: f

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(5,'suji','suji@gmail.com',18,'f')

1 row created.

### **DEFAULT CONSTRAINT**

SQL> create table stud(id number(2),name varchar(20),age number(2) default 18);

Table created.

SQL> insert into stud(id,name) values(1,'dinesh');

1 row created.

SQL> select \* from stud;

ID	NAME	AGE
1	dinesh	18

SQL> insert into stud(id,name,age) values(2,'gunal',19);

1 row created.

SQL> select \* from stud;

ID	NAME	AGE
1	dinesh	18
2	gunal	19

### **Foreign key:**

#### **Create the parent table(Department)**

```
SQL> create table department(deptname varchar(20) primary key,building varchar(15),budget number(10,2) );
```

Table created.

#### **Create the child table(Instructor)**

```
SQL> create table instructor(id varchar(20) primary key,name varchar(20),deptname varchar(20) references department(deptname));
```

Table created.

#### **Insert records into parent table(Department)**

```
SQL> insert into department values('&deptname','&building',&budget);
```

Enter value for deptname: IT

Enter value for building: seminar hall

Enter value for budget: 10000.00

```
old 1: insert into department values('&deptname','&building',&budget)
```

```
new 1: insert into department values('IT','seminar hall',10000.00)
```

1 row created.

```
SQL> /
```

Enter value for deptname: CSE

Enter value for building: seminar hall

Enter value for budget: 20000.00

```
old 1: insert into department values('&deptname','&building',&budget)
```

```
new 1: insert into department values('CSE','seminar hall',20000.00)
```

1 row created.

```
SQL> /
```

Enter value for deptname: ECE

Enter value for building: seminar hall

Enter value for budget: 30000.00

```
old 1: insert into department values('&deptname','&building',&budget)
```

```
new 1: insert into department values('ECE','seminar hall',30000.00)
```

1 row created.

SQL> /

Enter value for deptname: AIDS

Enter value for building: seminar hall

Enter value for budget: 40000.00

old 1: insert into department values('&deptname','&building','&budget')

new 1: insert into department values('AIDS','seminar hall',40000.00)

1 row created.

### **Insert records into child table (Instructor)**

SQL> insert into instructor values(&id,&name,&deptname);

Enter value for id: 1

Enter value for name: ram

Enter value for deptname: IT

old 1: insert into instructor values(&id,&name,&deptname')

new 1: insert into instructor values(1,'ram','IT')

1 row created.

SQL> /

Enter value for id: 2

Enter value for name: saai

Enter value for deptname: CSE

old 1: insert into instructor values(&id,&name,&deptname')

new 1: insert into instructor values(2,'saai','CSE')

1 row created.

SQL> /

Enter value for id: 3

Enter value for name: gunal

Enter value for deptname: ECE

old 1: insert into instructor values(&id,&name,&deptname')

new 1: insert into instructor values(3,'gunal','ECE')

1 row created.



**To see the instance of parent table:**

SQL> select \* from department;

DEPTNAME	BUILDING	BUDGET
IT	seminar hall	10000
CSE	seminar hall	20000
ECE	seminar hall	30000
AIDS	seminar hall	40000

**To see the instance of child table:**

SQL> select \* from instructor;

ID	NAME	DEPTNAME
1	ram	IT
2	saai	CSE
3	gunal	ECE

**Insert the instance into child table that is not in parent table:**

SQL> insert into instructor values(&id,&name','&deptname');

Enter value for id: 4

Enter value for name: babu

Enter value for deptname: CSD

old 1: insert into instructor values(&id,&name','&deptname')

new 1: insert into instructor values(4,'babu','CSD')

insert into instructor values(4,'babu','CSD')

\*

**ERROR at line 1:**

**ORA-02291: integrity constraint (STUDENT.SYS\_C004060) violated - parent key not**

**Found**

**Delete the instance in the parent table :**

```
SQL> delete from department where deptname='CSE';
```

```
delete from department where deptname='CSE'
```

\*

**ERROR at line 1:**

**ORA-02292: integrity constraint (STUDENT.SYS\_C004060) violated - child record**

**Found**

**Delete the instance in the child table:**

**Delete the records in table1:**

```
SQL> delete from instructor where deptname='CSE';
```

1 row deleted.

**To see the instance of child table**

```
SQL> select * from instructor;
```

ID	NAME	DEPTNAME
1	ram	IT
3	gunal	ECE

```
SQL> delete from department where deptname='CSE';
```

1 row deleted.

**To see the instance of parent table**

```
SQL> select * from department;
```

DEPTNAME	BUILDING	BUDGET
IT	seminar hall	10000
ECE	seminar hall	30000
AIDS	seminar hall	40000

### **DROP COMMAND:**

#### **Drop the parent table:**

SQL> drop table department;

drop table department

\*

#### **ERROR at line 1:**

**ORA-02449: unique/primary keys in table referenced by foreign keys**

#### **Drop the child table:**

SQL> drop table instructor;

Table dropped.

SQL> drop table department;

Table dropped.

### **ON DELETE CASCADE COMMAND:**

SQL> create table instructor(id varchar(20) primary key,name varchar(20),deptname varchar(20) references department(deptname) on delete cascade);

Table created.

#### **Insert records into table2:**

SQL> insert into instructor values(&id,'&name','&deptname');

Enter value for id: 1

Enter value for name: ram

Enter value for deptname: IT

old 1: insert into instructor values(&id,'&name','&deptname')

new 1: insert into instructor values(1,'ram','IT')

1 row created.

SQL> /

Enter value for id: 2

Enter value for name: Saai

Enter value for deptname: CSE

old 1: insert into instructor values(&id,'&name','&deptname')

new 1: insert into instructor values(2,'Saai','CSE')

1 row created.

SQL> /

Enter value for id: 3

Enter value for name: gunal

Enter value for deptname: ECE

old 1: insert into instructor values(&id,'&name','&deptname')

new 1: insert into instructor values(3,'gunal','ECE')

1 row created.

**To see the instance of parent table before deleting**

SQL> select \* from department;

DEPTNAME	BUILDING	BUDGET
----------	----------	--------

-----

IT	seminar hall	10000
ECE	seminar hall	30000
AIDS	seminar hall	40000

**To delete the instance of parent table:**

SQL> delete from department where deptname='CSE';

1 row deleted.

**To see the instance of parent table:**

SQL> select \* from department;

DEPTNAME	BUILDING	BUDGET
----------	----------	--------

-----

IT	seminar hall	10000
ECE	seminar hall	30000

### **ON DELETE SET NULL COMMAND:**

```
SQL> create table instructor(id varchar(20) primary key,name varchar(20),deptname  
varchar(20) references department(deptname) on delete set null);
```

Table created.

```
SQL> insert into instructor values(&id,&name,&deptname);
```

Enter value for id: 1

Enter value for name: ram

Enter value for deptname: IT

```
old 1: insert into instructor values(&id,&name,&deptname')
```

```
new 1: insert into instructor values(1,'ram','IT')
```

1 row created.

```
SQL> /
```

Enter value for id: 6

Enter value for name: suba

Enter value for deptname: IT

```
old 1: insert into instructor values(&id,&name,&deptname')
```

```
new 1: insert into instructor values(6,'suba','IT')
```

1 row created.

```
SQL> /
```

Enter value for id: 7

Enter value for name: sanjay

Enter value for deptname: ECE

```
old 1: insert into instructor values(&id,&name,&deptname')
```

```
new 1: insert into instructor values(7,'sanjay','ECE')
```

1 row created.

### **To delete the instance of parent table:**

```
SQL> delete from department where deptname='ECE';
```

1 row deleted.

SQL> select \* from department;

DEPTNAME	BUILDING	BUDGET
----------	----------	--------

IT	seminar hall	10000
----	--------------	-------

SQL> select \* from instructor;

ID	NAME	DEPTNAME
----	------	----------

1	ram	IT
---	-----	----

6	suba	IT
---	------	----

7	sanjay	
---	--------	--

**Truncate command:**

SQL> truncate table instructor;

Table truncated.

**To see the instance of child table**

SQL> select \* from instructor;

no rows selected

**To see the schema of the table:**

SQL> desc instructor;

Name	Null?	Type
------	-------	------

ID	NOT NULL	VARCHAR2(20)
----	----------	--------------

NAME		VARCHAR2(20)
------	--	--------------

DEPTNAME		VARCHAR2(20)
----------	--	--------------

**Rollback command:**

SQL> rollback;

Rollback complete.

SQL> select \* from instructor;

no rows selected

### To see the data dictionary

SQL> select \* from tab;

TNAME	TABTYPE	CLUSTERID
-------	---------	-----------

CUSTOMERS	TABLE	
DETAIL	TABLE	
EMP1	TABLE	
STUD	TABLE	
EMP	TABLE	
STUDENT	TABLE	
CUST	TABLE	
EMPLOYEE	TABLE	
EMPP	TABLE	
CUS	TABLE	
CUSTOM	TABLE	
CUSTOMER_DETAILS	TABLE	
DEPARTMENT	TABLE	
INSTRUCTOR	TABLE	

### ALTER COMMAND

#### ALTER WITH ADD COMMAND:

SQL> alter table department add(mailid varchar(20));

Table altered.

SQL> desc department;

Name	Null?	Type
DEPTNAME	NOT NULL	VARCHAR2(20)
BUILDING		VARCHAR2(15)
BUDGET		NUMBER(10,2)
<b>MAILID</b>		<b>VARCHAR2(20)</b>

### **ALTER WITH MODIFY COMMAND:**

```
SQL> alter table department modify(mailid varchar(25));
```

Table altered.

```
SQL> desc department;
```

Name	Null?	Type
-----		
DEPTNAME	NOT NULL	VARCHAR2(20)
BUILDING		VARCHAR2(15)
BUDGET		NUMBER(10,2)
MAILID		VARCHAR2(25)

### **To delete the column using alter command:**

```
SQL> alter table department drop column mailid;
```

Table altered.

```
SQL> desc department;
```

Name	Null?	Type
-----		
DEPTNAME	NOT NULL	VARCHAR2(20)
BUILDING		VARCHAR2(15)
BUDGET		NUMBER(10,2)

### **RENAME COMMAND:**

#### **Rename the parent table:**

```
SQL> rename department to dept;
```

Table renamed.

```
SQL> desc dept;
```

Name	Null?	Type
-----		
DEPTNAME	NOT NULL	VARCHAR2(20)
BUILDING		VARCHAR2(15)
BUDGET		NUMBER(10,2)



**To see the parent after renaming:**

SQL> desc department;

**ERROR:**

**ORA-04043: object department does not exist**

**DELETION:**

SQL> delete from dept;

4 rows deleted.

**RECORDS AFTER DELETION:**

SQL> select \* from dept;

no rows selected

**Rollback**

SQL> rollback;

Rollback complete.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
----------	----------	--------

IT	seminar hall	10000
----	--------------	-------

CSE	seminar hall	20000
-----	--------------	-------

ECE	seminar hall	30000
-----	--------------	-------

AIDS	seminar hall	40000
------	--------------	-------

SQL>delete from dept where budget=20000;

1 row deleted.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
----------	----------	--------

IT	seminar hall	10000
----	--------------	-------

ECE	seminar hall	30000
-----	--------------	-------

AIDS	seminar hall	40000
------	--------------	-------

SQL>delete from dept where deptname='AIDS';

1 row deleted.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
----------	----------	--------

IT	seminar hall	10000
ECE	seminar hall	30000

CONTENTS	MARKS ALLOTTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

## **RESULT**

Thus Data Definition Language commands and Integrity Constraints were executed.

**EX.NO:2                      DATA MANIPULATION LANGUAGE,DATA CONTROL  
DATE:07.03.2024                      LANGUAGE COMMANDS**

**AIM:**

To execute DML,DCL AND TCL commands using SQL.

**Create the Relation(table):**

create table student(roll number(10) primary key,name varchar(20) not null,mail varchar(20) unique,age number(3),gender char(1) check(gender='m' or gender='f'));

Table created.

**Inserting instance in table:**

insert into student values(&roll,&name,&mail,&age,&gender);

SQL> insert into student values(&roll,&name,&mail,&age,&gender);

Enter value for roll: 101

Enter value for name:dinesh

Enter value for mail: dinesh@gmail.com

Enter value for age: 18

Enter value for gender: f

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(101,'dinesh','dinesh@gmail.com',18,'f')

1 row created.

SQL> /

Enter value for roll: 102

Enter value for name: babu

Enter value for mail: babu@gmail.com

Enter value for age: 19

Enter value for gender: m

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(102,'babu','babu@gmail.com',19,'m')

1 row created.

SQL> /

Enter value for roll: 103

Enter value for name: ram

Enter value for mail: ram@gmail.com

Enter value for age: 19

Enter value for gender: m

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(103,'ram','ram@gmail.com',19,'m')

1 row created.

SQL> /

Enter value for roll: 104

Enter value for name: raj

Enter value for mail: raj@gmail.com

Enter value for age: 19

Enter value for gender: m

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(104,'raj','raj@gmail.com',19,'m')

1 row created.

SQL> /

Enter value for roll: 105

Enter value for name: gunal

Enter value for mail: gunal@gmail.com

Enter value for age: 20

Enter value for gender: f

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(105,'gunal','gunal@gmail.com',20,'f')

1 row created.

**To see the structure of the table:**

SQL> desc student;

Name	Null?	Type
-----	-----	-----
ROLL	NOT NULL	NUMBER(10)
NAME	NOT NULL	VARCHAR2(20)
MAIL		VARCHAR2(20)
AGE		NUMBER(3)
GENDER		CHAR(1)

**To see the instance of the table:**

SQL> select \* from student;

ROLL	NAME	MAIL	AGE	G
-----	-----	-----	-----	---
101	dinesh	dinesh@gmail.com	18	f
102	babu	babu@gmail.com	19	m
103	ram	ram@gmail.com	19	m
104	raj	raj@gmail.com	19	m
105	gunal	gunal@gmail.com	20	f

**Commit Command:**

SQL> commit;

Commit complete.

SQL> insert into student values(&roll,'&name','&mail',&age,'&gender');

Enter value for roll: 106

Enter value for name: hema

Enter value for mail: hema@gmail.com

Enter value for age: 19

Enter value for gender: f

old 1: insert into student values(&roll,'&name','&mail',&age,'&gender')

new 1: insert into student values(106,hema,'hema@gmail.com',19,'f')

1 row created.

SQL> /

Enter value for roll: 107

Enter value for name:saai

Enter value for mail: saai@gmail.com

Enter value for age: 19

Enter value for gender: m

old 1: insert into student values(&roll,&name,&mail,&age,&gender')

new 1: insert into student values(107,'saai','saai@gmail.com',19,'m')

1 row created.

SQL> select \* from student;

ROLL	NAME	MAIL	AGE	G
-----	-----	-----	-----	----
101	dinesh	dinesh@gmail.com	18	f
102	babu	babu@gmail.com	19	m
104	raj	raj@gmail.com	19	m
105	gunal	gunal@gmail.com	20	f
106	hema	hema@gmail.com	19	f
107	saai	saai@gmail.com	19	m

6 rows selected.

### **Rollback command:**

SQL> rollback;

Rollback complete.

SQL> select \* from student;

ROLL	NAME	MAIL	AGE	G
-----	-----	-----	-----	----
101	dinesh	dinesh@gmail.com	18	f
102	babu	babu@gmail.com	19	m
104	raj	raj@gmail.com	19	m

105      gunal                      gunal@gmail.com              20      f

**Select command:**

SQL> select \* from student;

ROLL	NAME	MAIL	AGE	G
-----	-----	-----	-----	----
101	dinesh	dinesh@gmail.com	18	f
102	babu	babu@gmail.com	19	m
104	raj	raj@gmail.com	19	m
105	gunal	gunal@gmail.com	20	f

SQL>select roll,name from student;

ROLL	NAME
-----	-----

101	dinesh
102	babu
104	raj
105	gunal

SQL>select name from student where roll=101;

NAME
-----
dinesh

**SAVEPOINT:-**

SQL> insert into student values('&roll','&name','&mail','&age','&gender');

Enter value for roll: 106

Enter value for name:hema

Enter value for mail:hema@gmail.com

Enter value for age: 19

Enter value for gender: f

old 1:insert into student values('&roll','&name','&mail','&age','&gender');

new 1: insert into student values('106','hema','hemai@gmail.com',19,'f,')

1 row created.

SQL> savepoint A;

Savepoint created.

SQL> insert into student values('&roll','&name','&mail','&age','&gender');

Enter value for roll: 107

Enter value for name:saai

Enter value for mail:saai@gmail.com

Enter value for age: 19

Enter value for gender: m

old 1:insert into student values('&roll','&name','&mail','&age','&gender');

new 1: insert into student values(107,'saai','saai@gmail.com,19,'m')

1 row created.

SQL> rollback to A;

Rollback complete.

SQL> select \* from student;

ROLL	NAME	MAIL	AGE	G
-----	-----	-----	-----	----
101	dinesh	dinesh@gmail.com	18	f
102	babu	babu@gmail.com	19	m
104	raj	raj@gmail.com	19	m
105	gunal	gunal@gmail.com	20	f
106	hema	hema@gmail.com	19	f

### **Update command:**

SQL> create table department(deptname varchar(20) primary key,building varchar(15),budget number(10,2) );

Table created.



**To Insert instance into table:**

SQL> insert into department values('&deptname','&building',&budget);

Enter value for deptname: IT

Enter value for building: seminar hall

Enter value for budget: 10000.00

old 1: insert into department values('&deptname','&building',&budget)

new 1: insert into department values('IT','seminar hall',10000.00)

1 row created.

SQL>select \* from department;

DEPTNAME	BUILDING	BUDGET
IT	seminar hall	10000

SQL> insert into dept(deptname,building) values('CSE','CC');

1 row created.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
IT	seminar hall	10000
CSE	CC	

**After update the budget of IT department:**

SQL> update dept set budget=100000.00 where budget=10000.00;

1 row updated.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
IT	seminar hall	100000
CSE	CC	

**Add missing records:**

SQL> update dept set budget=100000.00 where deptname='CSE';

1 row updated.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
-----	-----	-----
IT	seminar hall	100000
CSE	CC	100000

SQL> insert into dept(deptname) values('ECE');

1 row created.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
-----	-----	-----
IT	seminar hall	100000
CSE	CC	100000
ECE		

SQL> update dept set building='Lab',budget=200000.00 where deptname='ECE';

1 row updated.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
-----	-----	-----
IT	seminar hall	100000
CSE	CC	100000
ECE	Lab	200000

### **Update with case :**

SQL> create table emp(empid number(5),empname varchar(20),salary number(6));

Table created.

SQL> insert into emp values(&empid,&empname,&salary);

Enter value for empid: 1

Enter value for empname: dinesh

Enter value for salary: 15000

old 1: insert into emp values(&empid,&empname,&salary)

new 1: insert into emp values(1,'dinesh ',15000)

1 row created.

SQL> /

Enter value for empid: 2

Enter value for empname: Gunal

Enter value for salary: 17000

old 1: insert into emp values(&empid,&empname,&salary)

new 1: insert into emp values(2,'Gunal',17000)

1 row created.

SQL> /

Enter value for empid: 3

Enter value for empname: Anand

Enter value for salary: 10000

old 1: insert into emp values(&empid,&empname,&salary)

new 1: insert into emp values(3,'Anand',10000)

1 row created.

SQL> /

Enter value for empid: 4

Enter value for empname: Babu

Enter value for salary: 17000

old 1: insert into emp values(&empid,&empname,&salary)

new 1: insert into emp values(4,'Babu',17000)

1 row created.

SQL> /

Enter value for empid: 5

Enter value for empname: suji

Enter value for salary: 10000

old 1: insert into emp values(&empid,&empname,&salary)

new 1: insert into emp values(5,'suji',10000)

1 row created.

**Before updating the salary bonus:**

SQL> select \* from emp;

EMPID	EMPNAME	SALARY
-----	-----	-----
1	dinesh	15000
2	Gunal	17000
3	Anand	10000
4	Babu	17000
5	suji	10000

SQL> update emp set salary=case when salary <=10000 then salary\*1.03 else salary\*1.05 end;

5 rows updated.

**After updating the salary bonus:**

SQL> select \* from emp;

EMPID	EMPNAME	SALARY
-----	-----	-----
1	dinesh	15750
2	Gunal	17850
3	Anand	10300
4	Babu	17850
5	suji	10300

**SAVEPOINT:**

SQL> update dept set budget=150000 where deptname='IT';

1 row updated.

SQL> savepoint A;

Savepoint created.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
-----	-----	-----
IT	seminar hall	150000
CSE	CC	100000
ECE	Lab	200000

SQL> update dept set budget=125000 where deptname='IT';

1 row updated.

SQL> savepoint B;

Savepoint created.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
IT	seminar hall	125000
CSE	CC	100000
ECE	Lab	200000

SQL> rollback to A;

Rollback complete.

SQL> select \* from dept;

DEPTNAME	BUILDING	BUDGET
IT	seminar hall	150000
CSE	CC	100000
ECE	Lab	200000

SQL> savepoint A;

Savepoint created.

CONTENTS	MARKS ALLOTTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

## **RESULT**

Thus DML,DCL and TCL commands were executed successfully.

**Ex No:3**

## **JOIN OPERATIONS**

**28.03.24**

### **AIM**

To execute the various types of join.

### **JOINS**

Joins in SQL are commands which are used to combine rows from two or more table, based on / a related column between those tables.

### **TYPES**

- CROSS JOIN OR CARTESIAN PRODUCT
- INNER JOIN
- NATURAL JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN
- SELF JOIN
- EQUI JOIN

### **CREATE A TABLE**

```
SQL> create table instructor(id varchar(3),name varchar(7),deptname varchar(5),salary  
number(6));
```

Table created.

```
SQL> create table teaches(id varchar(3),courseid varchar(4),secidvarchar(4),sem  
number(1),year number(1));
```

Table created.

## 1) CROSS JOIN OR CARTESIAN PRODUCT:

A Cartesian product, or cross join, combines each row of one table with every row of another table, resulting in a potentially large combined table. It's used in SQL queries when no join condition is specified between the tables.

```
SQL> select * from instructor, teaches;
```

ID	NAME	DEPTN	SALARY	ID	COUR	SECI	SEM	YEAR
01	DINESH	IT	50000	01	432	IT1	4	2
02	GUNAL	CSE	40000	01	432	IT1	4	2
03	ANAND	ML	30000	01	432	IT1	4	2
04	ARUN	DS	30000	01	432	IT1	4	2
01	DINESH	IT	50000	02	532	CSE3	4	2
02	GUNAL	CSE	40000	02	532	CSE3	4	2
03	ANAND	ML	30000	02	532	CSE3	4	2
04	ARUN	DS	30000	02	532	CSE3	4	2
01	DINESH	IT	50000	03	675	ML3	4	2
02	GUNAL	CSE	40000	03	675	ML3	4	2
03	ANAND	ML	30000	03	675	ML3	4	2

04	ARUN	DS	30000	03	675	ML3	4	2
01	DINESH	IT	50000	04	654	DS4	4	2
02	GUNAL	CSE	40000	04	654	DS4	4	2
03	ANAND	ML	30000	04	654	DS4	4	2
04	ARUN	DS	30000	04	654	DS4	4	2

16 rows selected.

## 2) NATURAL JOIN

A natural join is an operation that combines tables based on columns with the same name and data type, producing a result that includes only the matching rows.

### NATURAL JOIN USING KEYWORD

SQL> select \* from instructor natural join teaches;

ID	NAME	DEPTN	SALARY	COUR	SECI	SEM	YEAR
01	DINESH	IT	50000	432	IT1	4	2
02	GUNAL	CSE	40000	532	CSE3	4	2
03	ANAND	ML	30000	675	ML3	4	2
04	ARUN	DS	30000	654	DS4	4	2

### NATURAL JOIN USING WHERE CONDITION

SQL> select \* from instructor, teaches where instructor.id=teaches.id;



ID	NAME	DEPTN	SALARY	ID	COUR	SECI	SEM	YEAR
----	------	-------	--------	----	------	------	-----	------

-----

01	DINESH	IT	50000	01	432	IT1	4	2
02	GUNAL	CSE	40000	02	532	CSE3	4	2
03	ANAND	ML	30000	03	675	ML3	4	2
04	ARUN	DS	30000	04	654	DS4	4	2

### 3) INNER JOIN

An inner join is a type of join operation that combines rows from two tables based on a related column between them. It returns only the rows that have matching values in both tables, according to the specified join condition.

SQL> select \* from instructor inner join teaches on instructor.id=teaches.id;

ID	NAME	DEPTN	SALARY	ID	COUR	SECI	SEM	YEAR
----	------	-------	--------	----	------	------	-----	------

-----

01	DINESH	IT	50000	01	432	IT1	4	2
02	GUNAL	CSE	40000	02	532	CSE3	4	2
03	ANAND	ML	30000	03	675	ML3	4	2
04	ARUN	DS	30000	04	654	DS4	4	2

## 1)LEFT OUTER JOIN

A left outer join is a join operation that combines rows from two tables based on a related column, preserving all rows from the lefttable and matching rows from the right table.

### LEFT OUTER JOIN USING KEYWORD

```
SQL> select*from emp natural left outer join cust;
```

CITY	EMPID	CUSTI
------	-------	-------

-----	-----	
-------	-------	--

CHICAGO	A4	B4
---------	----	----

CHICAGO	A3	B4
---------	----	----

NEW YORK	A1	
----------	----	--

NULL	A2	
------	----	--

PARIS	A5	
-------	----	--

### LEFT OUTER JOIN USING (+) SYMBOL

```
SQL> select*from emp,cust where emp.city=cust.city(+);
```

EMPID	CITY	CUSTI	CITY
-------	------	-------	------

-----			
-------	--	--	--

A4	CHICAGO	B4	CHICAGO
----	---------	----	---------

A3	CHICAGO	B4	CHICAGO
----	---------	----	---------

A1	NEW YORK		
----	----------	--	--

A2	NULL		
----	------	--	--

A5	PARIS		
----	-------	--	--

## 2)RIGHT OUTER JOIN

A right outer join is a join operation that combines rows from two tables based on a related column, preserving all rows from the righttable and matching rows from the left table.

### RIGHT OUTER JOIN USING KEYWORD

```
SQL> select*from emp natural right outer join cust;
```

CITY	EMPID	CUSTI
------	-------	-------

-----

CHICAGO	A3	B4
---------	----	----

MOSCOW		B5
--------	--	----

NEWYORK		B2
---------	--	----

NEWYORK		B1
---------	--	----

### RIGHT OUTER JOIN USING (+) SYMBOL

```
SQL> select*from emp,cust where emp.city(+)=cust.city;
```

EMPID	CITY	CUSTI	CITY
-------	------	-------	------

-----

A3	CHICAGO	B4	CHICAGO
----	---------	----	---------

	B5	MOSCOW	B2
--	----	--------	----

	NEWYORK		B1
--	---------	--	----

	NEWYORK		
--	---------	--	--

### 3)FULL OUTER JOIN

A full outer join is a join operation that combines rows from two tables based on a related column, preserving all rows from both tables.

```
SQL> select*from emp natural full outer join cust;
```

CITY	EMPID	CUSTI
------	-------	-------

-----	-----	
-------	-------	--

CHICAGO	A3	B4
---------	----	----

PARIS	A5	
-------	----	--

NEW YORK	A1	
----------	----	--

MOSCOW		B5
--------	--	----

NEWYORK		B2
---------	--	----

NEWYORK		B1
---------	--	----

6 rows selected.

### 4)SELF JOIN

A self join is a join operation where a table is joined with itself. It's useful when querying hierarchical data or comparing rows within the same table

```
SQL> select e.ename,m.ename from employee e,employee m wheree.mid=m.eid;
```

ENAME	ENAME
-------	-------

-----	
-------	--

Jakku      DINESH

Moni      Jakku

DINESH   GUNAL

GUNAL    Moni

## 5)EQUI JOIN

An equi join is a type of join operation that combines rows from twotables based on a matching condition specified in the join clause.

```
SQL> select * from instructor,teaches where instructor.id=teaches.idand  
instructor.salary>30000;
```

ID	NAME	DEPTN	SALARY	ID	COUR	SECI	SEM	YEAR
01	DINESH	IT	50000	01	432	IT1	4	2
02	GUNAL	CSE	40000	02	532	CSE3	4	2

CONTENTS	MARKS ALLOTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

## RESULT

Thus the various join operation are executed successfully

**EX NO : 04**  
**18.04.2024**

## **NESTED QUERIES**

### **AIM:**

To execute Nested Query using IN, NOT IN, Some, ALL, Exists and Not Exists keywords.  
Also to execute scalar sub query in select and insert clause.

### **TYPES:**

- Independent sub-query – execution of Inner query is independent of Outer query.
- Co-related sub-query - execution of Inner query is dependent of Outer query.
- Scalar sub query- selects only one column or expression and returns one value or one row.

### **To create a relation:**

```
SQL> create table student(Rollno number(6),Name varchar(5),Mark number(4),DID number(4));
```

Table created.

```
SQL> select * from student;
```

ROLLNO	NAME	MARK	DID
-----	-----	-----	-----
101	DINESH	57	2
102	GUNAL	47	1
103	LALIT	75	1
104	AANAND	85	3
105	ALLWIN	59	1

### **Independent sub-query:**

#### **Using in keyword**

Ex: Find the Rollno and Name of the student who got marks above 60

```
SQL> select Rollno,Name from student where Rollno in(select Rollno from student where mark>60);
```

ROLLNO	NAME
-----	-----

103     LALIT  
104     AANAND

### Using not in keyword

EX: Find the Rollno and Name of the student who got marks other than 60

SQL> select Rollno,Name from student where Rollno not in(select Rollno from student where mark=60);

ROLLNO	NAME
101	DINESH
102	GUNAL
103	LALIT
104	AANAND
105	ALLWIN

### Using some keyword

Ex: Find the names of students who have scored a mark greater than some other student's mark

SQL> select Name from student where mark>some(select mark from student);

NAME  
AANAND  
LALIT  
ALLWIN  
DINESH

### Using All keyword

Ex: Find the name of student who have scored second highest mark

SQL> select Name from student where mark<(select max(Mark) from student);

NAME  
LALIT

-----     select Name from student where mark<>(select max(mark) from student );

### **Using Exists keyword**

EX: Find the names of students who have received a DID value of 1:

SQL> select Name from student where exists (select Name from student where DID=1);

NAME

-----

GUNAL

LALIT

ALLWIN

### **Using not Exists keyword**

EX: Find the names of students who have a mark less than or equal to 60.

SQL> select name from student s where not exists (select name from student where Mark>60 and s.Rollno=Rollno);

NAME

-----

GUNAL

DINESH

ALLWIN

### **Correlated sub-query:**

Ex: Find the Name of the student who got marks above 60

SQL> select Name from student O where Mark=(select Mark from student I where I.Rollno=O.Rollno and Mark>60);

NAME

-----

LALIT

AANAND

### **Using Exists keyword**

Ex: Find the Rollno and Name of the student who got marks above 60



SQL> select Rollno,Name from student O where exists (select Mark from student I where O.Rollno=I.Rollno and Mark>60);

ROLLNO	NAME
--------	------

103	LALIT
104	AANAND

**Scalar Sub query:**

**Scalar subquery using select clause:**

SQL> create table department(did number(5),deptname varchar(20));

SQL> select \* from department;

DID	DEPTNAME
1	CSE
2	IT
3	ECE

SQL> create table faculty(fid number(4),fname varchar(20),deptname varchar(10));

SQL> select \* from faculty;

FID	FNAME	DEPTNAME
1	Dinesh	IT
2	Lalit	IT
3	Gunjal	CSE
4	Akhil	ECE
5	Aanand	ECE
6	Karthik	ECE

SQL> select deptname,(select count(\*) from faculty where faculty.deptname=department.deptname) as no\_of\_faculty from department;

DEPTNAME	NO_OF_FACULTY
CSE	1
IT	2

**Scalar subquery using insert clause:**

```
SQL> create table emp(emp_name varchar(10),dept_name varchar(10),salary
number(6),bonus number(5));
```

Table created.

```
SQL> select * from emp;
```

EMP_NAME	DEPT_NAME	SALARY	BONUS
Dinesh	IT	50000	5000
Gunal	CSE	40000	4500
Aanand	ECE	30000	4000
Allwin	AIDS	30000	3500
Praveen	CHEM	28000	3000

```
SQL> create table emp_salary_summary(sum_salaries number(6),max_salary
number(6),min_salary number(6),avg_salary number(6));
```

Table created.

**To see the records in emp\_salary\_summary:**

```
SQL>select * from emp_salary_summary;
```

no rows selected.

**Scalar subquery using insert clause:**

```
SQL> insert into emp_salary_summary(sum_salaries,max_salary,min_salary,avg_salary)
values( (select sum(salary) from emp),( select max(salary) from emp),(select min(salary)
from emp),(select avg(salary) from emp));
```

1 row created.

```
SQL> select * from emp_salary_summary;
```

SUM_SALARIES	MAX_SALARY	MIN_SALARY	AVG_SALARY
178000	50000	28000	35600

CONTENTS	MARKS ALLOTTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

**Result:**

Thus, the execution of Nested query using in, not in, some, all, exists and not exists key words, also the execution of scalar sub query using select and insert clause are executed successfully.

**Ex.No.5**

## **VIEWS AND INDEX**

**18.04.2024**

### **AIM:**

To create different views and different index methods using SQL commands.

#### **A) VIEWS**

Views are some kind of virtual tables created by original tables from the database.

#### **B) INDEX**

Indexes are special lookup tables that need to be used by the database search engine to speed up data retrieval.

### **TABLES:**

SQL> create table emp(eid number(5),name varchar(20),salary number(7),dno number(5));

SQL> select \* from emp;

EID	NAME	SALARY	DNO
1	Dinesh	70000	3
2	Gunal	60000	4
3	Aanand	60000	2
4	Lalit	75000	1
5	Karthik	50000	3
6	Akhil	55000	4

SQL> create table dept(eid number(5),dname varchar(20));

SQL> select \* from dept;

EID	DNAME
1	IT
2	CSE
3	ECE
4	AIDS
5	CSD
6	FT

SQL> create table faculty(fid number(5) primary key,fname varchar(20),dept varchar(5),salary number(7));

SQL> select \* from faculty;

FID	FNAME	DEPT	SALARY
1	Dinesh	IT	80000
2	Gunal	CSE	70000
3	Aanand	ECE	65000
4	Karthik	CSD	60000
5	Akhil	AIDS	55000

### **A)VIEWS**

#### **1. Creating the view by selecting the attributes from one table.**

SQL> create view empview as select eid,name,salary,dno from emp;

View created.

SQL> select \* from empview;

EID	NAME	SALARY	DNO
1	Dinesh	70000	3
2	Gunal	60000	4
3	Aanand	60000	2
4	Lalit	75000	1
5	Karthik	50000	3
6	Akhil	55000	4

6 rows selected.

#### **2. Creating the view by selecting the attributes from one table with predicate**

SQL> create view evview as select eid,name,salary,dno from emp where dno=3 or dno=4;

View created.

SQL> select \* from eviiew;

EID NAME	SALARY	DNO
-----		
1 Dinesh	70000	3
2 Gunal	60000	4
5 Karthik	50000	3
6 Akhil	55000	4

### 3. Creating the view by selecting the attributes from two tables with predicate

SQL> create view combo as select emp.eid as emp\_eid,name,salary,dno,dept.eid as dept\_eid,dname from emp,dept where emp.eid=dept.eid;

View created.

SQL> select \* from combo;

EMP_EID NAME	SALARY	DNO	DEPT_EID
-----			
1 Dinesh	70000	3	1
2 Gunal	60000	4	2
3 Aanand	60000	2	3
4 Lalit	75000	1	4
5 Karthik	50000	3	5
6 Akhil	55000	4	6

### 4. Creating the view using join clause.

SQL> create view emp2view as select e.eid,e.name,e.salary,e.dno,d.dname from emp e join dept d on e.eid=d.eid;

View created.

SQL> select \* from emp2view;

EID NAME	SALARY	DNO	DNAME
-----			
1 Dinesh	70000	3	IT

2 Gunal	60000	4	CSE
3 Aanand	60000	2	ECE
4 Lalit	75000	1	AIDS
5 Karthik	50000	3	CSD
6 Akhil	55000	4	FT

### 5. Creating the view using Sub query.

```
SQL> create view faculty_view as select f.fid,(select f.fname from faculty ff where
ff.fid=f.fid)as name from faculty f;
```

View created.

```
SQL> select * from faculty_view;
```

FID	NAME
1	Dinesh
2	Gunjal
3	Aanand
4	Karthik
5	Akhil

### 6. Creating the view using aggregate and group by clause.

```
SQL> create view faculty_view2(dept,total_salary)as select dept,sum(salary)
from faculty group by dept;
```

View created.

```
SQL> select * from faculty_view2;
```

DEPT	TOTAL_SALARY
IT	80000
CSD	60000
CSE	70000
AIDS	55000
ECE	65000

### 7.Creating the Materialized view.

```
SQL> create view fview as select f.dept,count(f.fname) as viewtable from faculty f group by f.dept;
```

View created.

```
SQL> select * from faculty;
```

FID	FNAME	DEPT	SALARY
1	Dinesh	IT	80000
2	Gunal	CSE	70000
3	Aanand	ECE	65000
4	Karthik	CSD	60000
5	Akhil	AIDS	55000

```
SQL> select * from fview;
```

DEPT	VIEWTABLE
IT	1
CSD	1
CSE	1
AIDS	1
ECE	1

```
SQL> insert into faculty values(6,'Allwin','IT',75000);
```

1 row created.

```
SQL> select * from faculty;
```

FID	FNAME	DEPT	SALARY
1	Dinesh	IT	80000
2	Gunal	CSE	70000
3	Aanand	ECE	65000
4	Karthik	CSD	60000



5 Akhil	AIDS	55000
---------	------	-------

6 Allwin	IT	75000
----------	----	-------

6 rows selected.

SQL> select \* from fview;

DEPT VIEWTABLE

-----

IT	2
----	---

CSD	1
-----	---

CSE	1
-----	---

AIDS	1
------	---

ECE	1
-----	---

### 8.Creating the Updatable view.

i)

SQL> create view f2view as select fid,fname from faculty;

View created.

SQL> select \* from f2view;

FID FNAME

-----

1 Dinesh
----------

2 Gunal
---------

3 Aanand
----------

4 Karthik
-----------

5 Akhil
---------

6 Allwin
----------

SQL> insert into f2view values(7,'Siva');

1 row created.

SQL> select \* from f2view;

FID FNAME

-----

1 Dinesh
----------

- 2 Gunal
- 3 Aanand
- 4 Karthik
- 5 Akhil
- 6 Allwin
- 7 Siva

SQL> select \* from faculty;

FID	FNAME	DEPT	SALARY
-----			
1	Dinesh	IT	80000
2	Gunal	CSE	70000
3	Aanand	ECE	65000
4	Karthik	CSD	60000
5	Akhil	AIDS	55000
6	Allwin	IT	75000
7	Siva		

7 rows selected.

ii)

SQL> create view emview as select e.eid,e.name,e.salary,e.dno,d.dname from emp e join dept d on e.eid=d.eid;

View created.

SQL> select \* from emview;

EID	NAME	SALARY	DNO	DNAME
-----				
1	Dinesh	70000	3	IT
2	Gunal	60000	4	CSE
3	Aanand	60000	2	ECE
4	Lalit	75000	1	AIDS
5	Karthik	50000	3	CSD
6	Akhil	55000	4	FT

6 rows selected.

SQL> select \* from emp;

EID	NAME	SALARY	DNO
1	Dinesh	70000	3
2	Gunal	60000	4
3	Aanand	60000	2
4	Lalit	75000	1
5	Karthik	50000	3
6	Akhil	55000	4

6 rows selected.

SQL> select \* from dept;

EID	DNAME
1	IT
2	CSE
3	ECE
4	AIDS
5	CSD
6	FT

6 rows selected.

SQL> insert into emview values(7,'Suresh',80000,5,'AIML');

insert into emview values(7,'Suresh',80000,5,'AIML')

\*

ERROR at line 1:

ORA-01779: cannot modify a column which maps to a non key-preserved table

**Reason:**

The view must be created on a single table. whereas here we created a view from two table.

## 9. Dropping the view

```
SQL> drop view f2view;
```

View dropped.

```
SQL> select * from f2view;
```

```
select * from f2view
```

```
      *
```

ERROR at line 1:

ORA-00942: table or view does not exist

## 10. Find Read only view or materialized view.

```
SQL> select view_name,case when view_type='MATERIALIZED VIEW' then 'Materialized  
view' else 'Regular View' end as view_type from all_views where  
view_name='FACULTY_VIEW2';
```

VIEW_NAME	VIEW_TYPE
-----------	-----------

-----

FACULTY_VIEW2	Regular View
---------------	--------------

### To see the all views in a directory:

```
select owner,view_name from all_views;
```

## **B)INDEX**

### 1. Single column Index :

```
SQL> create index teacher on faculty(fname);
```

Index created.

### 2. Composite index :

```
SQL> create index fac_detail on faculty(fname,dept);
```

Index created.

### 3. Dropping an Index:

```
SQL> drop index teacher;
```

Index dropped.

CONTENTS	MARKS ALLOTTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

### **Result**

Thus, the different views and different indexes were created using sql commands.

**EX.NO:06**

## **SQL STRING FUNCTIONS/OPERATIONS**

**26.04.2024**

### **AIM:**

To perform SQL string operations by using functions.

### **STRING OPERATIONS USING DUAL TABLE:**

#### **1.ASCII(CHR):**

SQL> select ascii('D') as ASCII\_OF\_D from dual;

ASCII\_OF\_D

-----

68

#### **2. CHR(NUMBER):**

SQL> select chr(68) from dual;

D

#### **3. CONCAT(string1,string2):**

SQL> select concat('DINESH','KUMAR') as concat from dual;

CONCAT

-----

DINESHKUMAR

#### **4.UPPER(STR):**

SQL> select upper('dinesh') as UPPER from dual;

UPPER

-----

DINESH

#### **5.LOWER(STR):**

SQL> select lower('DINESHKUMAR') as LOWER from dual;

LOWER

-----

dineshkumar

#### **6.LENGTH(STR):**

SQL> select length('DINESHKUMAR') as LENGTH from dual;

LENGTH

-----

11

#### **7. TRIM(STR):**

SQL> select trim(' Dineshkumar ') as TRIM from dual;

TRIM

-----

Dineshkumar

#### **8.REPLACE(STR):**

SQL> select replace('DINESHKUMAR VELMURUGAN','VELMURUGAN','V') as  
REPLACE fr

om dual;

REPLACE

-----

DINESHKUMAR V

#### **9.SUBSTR():**

SQL> select substr('DINESHKUMAR VELMURUGAN',1,11) as SUBSTR from dual;

SUBSTR

-----

DINESHKUMAR

#### **10.RPAD():**

SQL> select rpad('DINESH',4) as RPAD from dual;

RPAD

----

DINE

### **STRING OPERATIONS USING EMPLOYEE TABLE:**

#### **EMPLOYEE TABLE:**

SQL> select \* from employee;

EID	ENAME	SALARY	DEPT
1	Dinesh	20000	ECE
2	GUNAL	15000	CSE
3	Aanand	12000	IT
4	Diva	11000	ECE
5	Nadin	10000	IT

#### **1.CONCATENATION:**

SQL> select ename || ' works at ' || dept as emp\_details from employee;

EMP\_DETAILS

Dinesh works at ECE  
GUNAL works at CSE  
Aanand works at IT  
Diva works at ECE  
Nadin works at IT

#### **2.UPPER:**

SQL> select upper(ename) as ename\_upper from employee;

ENAME\_UPPER



DINESH  
GUNAL  
AANAND  
DIVA  
NADIN

### 3.LOWER:

SQL> select lower(ename) as ename\_lower from employee;

ENAME\_LOWE

-----

dinesh  
gunal  
aanand  
diva  
nadin

### 4.LENGTH OF STRING:

SQL> select length(ename) as ename\_length from employee;

ENAME\_LENGTH

-----

6  
5  
6  
4  
5

### 5.TRIM:

SQL> select trim(ename) as ename\_trimmed from employee;

ENAME\_TRIM

-----

Dinesh  
GUNAL

Aanand

Diva

Nadin

#### 6.REPLACE:

SQL> select replace(ename,'GUNAL','Gunal') as ename\_replaced from employee where eid=2;

ENAME\_REPLACED

Gunal

#### 7.INITCAP:

SQL> select initcap(ename)as ename\_initcap from employee;

ENAME\_INIT

-----

Dinesh

Gunal

Aanand

Diva

Nadin

#### 8.INSTR:

SQL> select instr(ename,'Diva') as position\_of\_Diva from employee;

POSITION\_OF\_DIVA

-----

0

0

0

1

0

#### 9.LEFT PAD:

```
SQL> select lpad(ename,5,'*') as ename_padded from employee;
```

ENAME

-----

Dines

GUNAL

Aanan

\*Diva

Nadin

#### 10.RPAD:

```
SQL> select rpad(ename,5,'*') as ename_padded from employee where eid=1;
```

ENAME

-----

Dines

#### 11.REVERSE:

```
SQL> select reverse(ename) as ename_reversed from employee where eid=2;
```

ENAME\_REVE

-----

LANUG

#### **STRING OPERATIONS RELATED TO EMAIL – REGEX:**

```
SQL> select * from employee;
```

EID	ENAME	SALARY	DEPT	EMAIL
-----	-------	--------	------	-------

-----

1	Dinesh	20000	ECE	dinesh@gmail.com
---	--------	-------	-----	------------------

2	GUNAL	15000	CSE	gunal@gmail.com
---	-------	-------	-----	-----------------

3	Aanand	12000	IT	aanand@kongu.edu
---	--------	-------	----	------------------

4	Diva	11000	ECE	diva@gmail.com
---	------	-------	-----	----------------

5	Nadin	10000	IT	nadin@kongu.edu
---	-------	-------	----	-----------------

#### 1. EXTRACTING THE DOMAIN NAME OF EMAIL:

```
SQL> select ename,substr(email,instr(email,'@')+1) as email_domain from empl  
oyee;
```

ENAME	EMAIL_DOMAIN
-------	--------------

Dinesh	gmail.com
--------	-----------

GUNAL	gmail.com
-------	-----------

Aanand	kongu.edu
--------	-----------

Diva	gmail.com
------	-----------

Nadin	kongu.edu
-------	-----------

## 2.CHECK FOR VALID EMAIL FORMAT:

```
SQL> SELECT eid,ename,email FROM employee WHERE REGEXP_LIKE(EMAIL, '^[A-  
Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');
```

EID	ENAME	EMAIL
-----	-------	-------

1	Dinesh	dinesh@gmail.com
---	--------	------------------

2	GUNAL	gunal@gmail.com
---	-------	-----------------

3	Aanand	aanand@kongu.edu
---	--------	------------------

4	Diva	diva@gmail.com
---	------	----------------

5	Nadin	nadin@kongu.edu
---	-------	-----------------

## 3. CHANGING DOMAIN NAME:

```
SQL> SELECT REPLACE(EMAIL, SUBSTR(EMAIL, INSTR(EMAIL, '@')), '@zoo.com')  
AS new_email FROM employee;
```

NEW_EMAIL
-----------

dinesh@zoo.com
----------------

gunal@zoo.com
---------------

aanand@zoo.com
----------------

diva@zoo.com
--------------

[nadin@zoo.com](mailto:nadin@zoo.com)

**4.COUNTING NO.OF.EMPLOYEES BASED ON DOMAIN NAME:**

SQL> select substr(email,instr(email,'@')+1) as email\_domain,count(\*) as emp

loyee\_count from employee group by substr(email,instr(email,'@')+1);

EMAIL_DOMAIN	EMPLOYEE_COUNT
--------------	----------------

-----

gmail.com	3
-----------	---

kongu.edu	2
-----------	---

**SEARCHING OPERATIONS USING LIKE AND NOT LIKE KEYWORD:**

SQL> select \* from employee where ename like'D%';

EID	ENAME	SALARY	DEPT	EMAIL
-----	-------	--------	------	-------

-----

1	Dinesh	20000	ECE	dinesh@gmail.com
---	--------	-------	-----	------------------

4	Diva	11000	ECE	diva@gmail.com
---	------	-------	-----	----------------

SQL> select \* from employee where ename not like'D%';

EID	ENAME	SALARY	DEPT	EMAIL
-----	-------	--------	------	-------

-----

2	GUNAL	15000	CSE	gunal@gmail.com
---	-------	-------	-----	-----------------

3	Aanand	12000	IT	aanand@kongu.edu
---	--------	-------	----	------------------

5	Nadin	10000	IT	nadin@kongu.edu
---	-------	-------	----	-----------------

SQL> select \* from employee where ename like '%h';

EID	ENAME	SALARY	DEPT	EMAIL
-----	-------	--------	------	-------

-----

1 Dinesh      20000 ECE    dinesh@gmail.com

SQL> select \* from employee where ename like'\_\_n%';

EID ENAME      SALARY DEPT    EMAIL

-----

1 Dinesh      20000 ECE    dinesh@gmail.com

3 Aanand      12000 IT      aanand@kongu.edu

SQL> select \* from employee where ename not like '%n%';

EID ENAME      SALARY DEPT    EMAIL

-----

2 GUNAL      15000 CSE    gunal@gmail.com

4 Diva      11000 ECE    diva@gmail.com

CONTENTS	MARKS ALLOTTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

### **RESULT:**

Thus, the SQL string operations are performed successfully.

**EX.NO: 07**

**Set Operation , Aggregate function & group by**

**26.04.2024**

**Aim:**

To perform aggregation on the relational database and group by records based on the conditions and to perform set operations.

**Tables:**

SQL> create table depositor(cname varchar(10),dep\_id number(7),accno varchar(6));

Table created.

SQL> select \* from depositor;

CNAME	DEP_ID	ACCNO
-------	--------	-------

-----

John	1001	AC01
------	------	------

Sita	1002	AC02
------	------	------

Vijay	1003	AC10
-------	------	------

Ram	1004	Ac14
-----	------	------

SQL> select \* from loan;

CNAME	LOANNO	ACCNO
-------	--------	-------

-----

John	2001	AC01
------	------	------

Reema	2002	AC03
-------	------	------

Manju	2003	AC11
-------	------	------

Vijay	2004	AC10
-------	------	------

**SET OPERATIONS:**

**Union**

SQL> select cname from depositor union select cname from loan;

CNAME
-------

-----

John
------

Manju
-------

Ram
-----

Reema

Sita

Vijay

6 rows selected.

### **Intersection**

SQL> select cname from depositor intersect select cname from loan;

CNAME

-----

John

Vijay

### **Except or minus**

SQL> select cname from depositor minus select cname from loan;

CNAME

-----

Ram

Sita

SQL> select cname from loan minus select cname from depositor;

CNAME

-----

Manju

Reema

SQL> create table employee(eid number(5) primary key,ename varchar(10),salary  
number(7),dept varchar(6));

Table created.

SQL> select \* from employee;

EID	ENAME	SALARY	DEPT
1	Dinesh	20000	ECE



2	Gunal	15000	CSE
3	Aanand	12000	IT
4	Diva	11000	ECE
5	Nadin	10000	IT

### **AVG:**

```
SQL> select avg(salary) as avg_salary from employee;
```

AVG\_SALARY

-----

13600

### **MAX:**

```
SQL> select max(salary) as max_salary from employee;
```

MAX\_SALARY

-----

20000

### **MIN:**

```
SQL> select min(salary) as min_salary from employee;
```

MIN\_SALARY

-----

10000

### **SUM:**

```
SQL> select sum(salary) as total_salary from employee;
```

TOTAL\_SALARY

-----

68000

### **COUNT:**

```
SQL> select count(eid) as no_of_employee from employee;
```

NO\_OF\_EMPLOYEE

-----

5

**Aggeregate function with groupby,and having.**

SQL> select \* from employee;

EID	ENAME	SALARY	DEPT
-----	-------	--------	------

-----

1	Dinesh	20000	ECE
2	GUNAL	15000	CSE
3	Aanand	12000	IT
4	Diva	11000	ECE
5	Nadin	10000	IT

SQL> select dept,avg(salary) from employee group by dept;

DEPT	AVG(SALARY)
------	-------------

-----

IT	11000
CSE	15000
ECE	15500

SQL> select dept,avg(salary) from employee group by dept having avg(salary)>12000;

DEPT	AVG(SALARY)
------	-------------

-----

CSE	15000
ECE	15500

**Distinct:**

SQL> select distinct dept from employee;

DEPT

-----  
IT

CSE

ECE

**Orderby:**

**Descending:**

SQL> select \* from employee order by salary desc;

EID	ENAME	SALARY	DEPT
1	Dinesh	20000	ECE
2	GUNAL	15000	CSE
3	Aanand	12000	IT
4	Diva	11000	ECE
5	Nadin	10000	IT

**Ascending:**

SQL> select \* from employee order by salary asc;

EID	ENAME	SALARY	DEPT
5	Nadin	10000	IT
4	Diva	11000	ECE
3	Aanand	12000	IT
2	GUNAL	15000	CSE
1	Dinesh	20000	ECE

SQL> select \* from employee order by salary;

EID	ENAME	SALARY	DEPT
5	Nadin	10000	IT
4	Diva	11000	ECE
3	Aanand	12000	IT
2	GUNAL	15000	CSE

1	Dinesh	20000	ECE
---	--------	-------	-----

**Orderby using name:**

SQL> select \* from employee order by ename;

EID	ENAME	SALARY	DEPT
3	Aanand	12000	IT
1	Dinesh	20000	ECE
4	Diva	11000	ECE
2	GUNAL	15000	CSE
5	Nadin	10000	IT

SQL> select ename,salary from employee order by ename asc,salary desc;

ENAME	SALARY
Aanand	12000
Dinesh	20000
Diva	11000
GUNAL	15000
Nadin	10000
nadin	18000
nadin	13000

SQL> select ename,salary from employee order by ename desc,salary asc;

ENAME	SALARY
nadin	13000
nadin	18000
Nadin	10000
GUNAL	15000

Diva	11000
Dinesh	20000
Aanand	12000

CONTENTS	MARKS ALLOTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

**RESULT:**

Thus, aggregation on the relational database and group by records based on the conditions and perform set operations were performed.

**EX.NO:08**

## **DATE AND TIME FUNCTIONS**

**02.05.2024**

### **AIM**

To perform SQL date and time functions.

### **Table:**

```
create table employee(eid number(5),ename varchar(10),salary number(7),dept varchar(6),doj date,dob date);
```

```
SQL> insert into empww values(1,'Dinesh',20000,'ECE',to_date('23-07-2002','dd-mm-yyyy'),to_date('23-07-1967','dd-mm-yyyy'));
```

```
SQL> select * from employee;
```

EID	ENAME	SALARY	DEPT	DOJ	DOB
1	Dinesh	20000	ECE	23-JUL-02	23-JUN-67
2	GUNAL	15000	CSE	12-OCT-00	01-MAR-55
3	Aanand	12000	IT	03-APR-96	30-MAR-40
4	Diva	11000	ECE	20-AUG-04	18-MAY-80
5	Nadin	10000	IT	30-NOV-08	27-DEC-99
6	nadin	13000	CSD	31-OCT-18	19-SEP-90
7	nadin	18000	FT	25-JUN-22	04-NOV-02

### **DATE AND TIME FUNCTIONS ON EMPLOYEE TABLE:**

#### **To Calculate experience for eid=1:**

```
SQL> select ename,dept,round(months_between(sysdate,doj)/12) as experience from employee where eid=1;
```

ENAME	DEPT	EXPERIENCE
Dinesh	ECE	22

**To Calculate age for eid=1:**

SQL> select ename,dept,round(months\_between(sysdate,dob)/12) as Age from employee  
where eid=1;

ENAME	DEPT	AGE
-----		
Dinesh	ECE	57

**To Calculate both age and experience for the relation:**

SQL> select ename,dept,round(months\_between(sysdate,dob)/12) as  
Age,round(months\_between(sysdate,doj)/12) as experience from employee;

ENAME	DEPT	AGE	EXPERIENCE
-----			
Dinesh	ECE	57	22
GUNAL	CSE	69	24
Aanand	IT	84	28
Diva	ECE	44	20
Nadin	IT	24	15
nadin	CSD	34	5
nadin	FT	21	2

**TO FIND THE PROMOTION DATE:**

SQL> select eid,ename,dept,doj+interval'10' year as promotion from employee;

	EID	ENAME	DEPT	PROMOTION
-----				
1	Dinesh	ECE	23-JUL-12	
2	GUNAL	CSE	12-OCT-10	
3	Aanand	IT	03-APR-06	
4	Diva	ECE	20-AUG-14	
5	Nadin	IT	30-NOV-18	

6 nadin	CSD	31-OCT-28
7 nadin	FT	25-JUN-32

### TO FIND YEAR OF JOINING:

SQL> select eid,ename,dept,extract(year from doj) as Year\_of\_joining from employee;

EID	ENAME	DEPT	YEAR_OF_JOINING
1 Dinesh	ECE		2002
2 GUNAL	CSE		2000
3 Aanand	IT		1996
4 Diva	ECE		2004
5 Nadin	IT		2008
6 nadin	CSD		2018
7 nadin	FT		2022

### To find Number of months working months:

SQL> select eid,ename,dept,trunc(months\_between(sysdate,doj)) as no\_of\_working\_months from employee;

EID	ENAME	DEPT	NO_OF_WORKING_MONTHS
1 Dinesh	ECE		261
2 GUNAL	CSE		282
3 Aanand	IT		336
4 Diva	ECE		236
5 Nadin	IT		185
6 nadin	CSD		66
7 nadin	FT		22

### TO FIND SENIOR EMPLOYEE :

SQL> select eid,ename,dept,doj from employee where doj=(select min(doj) from employee);

EID	ENAME	DEPT	DOJ
-----	-------	------	-----



-----  
3 Aanand IT 03-APR-96

**TO FIND EMPLOYEES HIRED ON SAME DAY:**

SQL> select doj,count(\*) as employees\_hired\_on\_same\_day from employee group by doj  
order by doj;

DOJ EMPLOYEES\_HIRED\_ON\_SAME\_DAY

-----  
03-APR-96 1  
12-OCT-00 1  
23-JUL-02 1  
20-AUG-04 1  
30-NOV-08 1  
31-OCT-18 1  
25-JUN-22 1

**TIMESTAMP:**

SQL>create table student(sid number(6) primary key,fname varchar(10),lname  
varchar(10),email varchar(20),enrollment\_date timestamp default current\_timestamp);

SQL> insert into student(sid,fname,lname,email) values(1,'john','Doe','john@  
gmail.com');

SQL> select \* from student;

SID	FNAME	LNAME	EMAIL	ENROLLMENT_DATE
-----	-------	-------	-------	-----------------

-----  
1 john Doe [john@gmail.com](mailto:john@gmail.com) 02-MAY-24 06.35.06.598000 PM

SQL>insert into student(sid,fname,lname,email,enrollment\_date)  
values(2,'dinesh','kumar','dineshkuma@gmail.com',to\_date('03-MAY-2024 10:30:00','DD-  
MON-YYYY HH24:MI:SS'));

SQL> select \* from student;

SID	FNAME	LNAME	EMAIL	ENROLLMENT_DATE
-----	-------	-------	-------	-----------------

-----  
1 john Doe [john@gmail.com](mailto:john@gmail.com) 02-MAY-24 06.35.06.598000 PM  
2 dinesh kumar [dineshkuma@gmail.com](mailto:dineshkuma@gmail.com) 03-MAY-24 10.30.00.000000 AM

## **DATE AND TIME FUNCTIONS ON DUAL :**

### **CURRENT DATE:**

\*This function is used to get the current date in the session time zone

```
SQL> select current_date from dual;
```

CURRENT\_DATE

-----

26-APR-24

### **SYS DATE:**

\*This function returns the current date and time of the Operating system

```
SQL> select sysdate from dual;
```

SYSDATE

-----

26-APR-24

### **EXTRACT:**

\* This extract function is used to retrieve a specific component which can be year, day, month.

#### **For Year**

```
SQL> select extract(year from to_date('2020-01-13','YYYY-MM-DD')) as YEAR from dual;
```

YEAR

-----

2020

#### **For Month**

```
SQL> select extract(month from to_date('2020-01-13','YYYY-MM-DD')) as MONTH from dual;
```

MONTH

-----

**TO\_DATE:**

\*This function converts a date which is in string type to date value

```
SQL> select to_date('23 JUL 2005','DD MON YYYY') converted_date from dual;
```

CONVERTED

-----

23-JUL-05

**TO\_CHAR:**

\* It is used to convert a date from DATE value to a specified date format.

```
SQL> select to_char(sysdate,'DD MM YYYY') as NEW_DATE from dual;
```

NEW\_DATE

-----

26 04 2024

```
SQL> select to_char(sysdate,'DD/MM/YYYY') as NEW_DATE from dual;
```

NEW\_DATE

-----

26/04/2024

**LAST\_DAY:**

\*This function is used to return the last day of the month of the particular date.

```
SQL> select last_day(sysdate) LAST_DAY from dual;
```

LAST\_DAY

-----

30-APR-24

**MONTHS\_BETWEEN:**

\*This function is used to calculate the months between two dates.

\* Round function is used to rounds the value to the specified decimal place.

```
SQL> select round(months_between(sysdate,date '2011-04-02')) MONTH_DIFFERENC
```

E from dual;

MONTH\_DIFFERENCE

-----

**ADD\_MONTHS:**

\*This function adds N months to a date and returns the same day N month after.

```
SQL> select add_months(sysdate,2) NEWDATE from dual;
```

NEWDATE

-----

26-JUN-24

**FROM\_TZ:**

\*This function converts the TIMESTAMP to TIMESTAMP with TIME ZONE value.

```
SQL> select from_tz(timestamp '2020-05-01 19:35:10','-07:00') NEWVALUE from  
dual;
```

NEWVALUE

-----

01-MAY-20 07.35.10.000000000 PM -07:00

**NEW\_TIME:**

\*This function converts a date from one time zone to a different time zone.

```
SQL> select new_time(sysdate,'pst','ast') TIME_IN_AST from dual;
```

TIME\_IN\_AST

-----

26-APR-24

**SESSIONTIMEZONE:**

\*This function as the name suggest returns the time zone of the current working session.

```
SQL> select sessiontimezone from dual;
```

SESSIONTIMEZONE

-----

+05:30

### **SYSTIMESTAMP:**

\*This function represents a timestamp with a time zone. It displays the result up to fractional seconds.

```
SQL> select systimestamp from dual;
```

SYSTIMESTAMP

-----

26-APR-24 04.14.19.952000 PM +05:30

### **TRUNC:**

\* TRUNC function in Oracle to truncate the current date (SYSDATE) to the beginning of the current month

```
SQL> select trunc(sysdate,'MM') MONTH from dual;
```

MONTH

-----

01-APR-24

### **TZ\_OFFSET:**

\*This function returns offset of a time zone name from UTC.

```
SQL> select TZ_OFFSET('Indian/Christmas') as OFFSET from dual;
```

OFFSET

-----

+07:00

### **To retrieve list of distinct time Zone:**

\*used to retrieve a list of distinct time zone names from the v\$timezone\_names view in Oracle database

```
SQL> select distinct tzname from v$timezone_names order by tzname;
```

CONTENTS	MARKS ALLOTTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

**RESULT:**

Thus, the SQL date and time functions were performed successfully.

EX.NO : 09

## PL/SQL STATEMENTS

09.05.2024

### **AIM :**

To execute PL/SQL statements in various programs.

### **PL/SQL:**

PL/SQL is a combination of SQL along with the procedural features of programming languages. PL/SQL is one of the key programming languages embedded in the Oracle Database along with SQL.

SQL> set serveroutput on

#### **1. Program to print Hello world**

SQL> declare

2 message varchar2(20):='Hello,world';

3 begin

4 dbms\_output.put\_line(message);

5 end;

6 /

Hello,world

PL/SQL procedure successfully completed.

#### **2. Program to print sum of two numbers**

SQL> declare

2 a integer:=0;

3 b integer:=20;

4 c integer;

5 f real;

6 begin

7 c:=a+b;

8 dbms\_output.put\_line('Value of c:'||c);

9 f:=70.0/3.0;

10 dbms\_output.put\_line('Value of f:'||f);

11 end;

12 /

Value of c:20

Value of f:23.33333333333333333333333333333333

PL/SQL procedure successfully completed.

### 3.Simple IF-THEN Statement

```
SQL> declare
```

2 n number;

3 begin

```
4  n:=&n;
```

5 if  $n > 0$  then

```
6 dbms_output.put_line('Given number is Greater than Zero');
```

```

7 end if;

```

8 end;

9 /

Enter value for n: 789

old 4:  $n := \&n;$

```
new 4: n:=789;
```

Given number is Greater than Zero

PL/SQL procedure successfully completed.

#### 4. Simple IF-THEN-ELSE Statement

```
SQL> declare
```

2 n number;

3 begin

4  $n := \&n;$

5 if  $n > 0$  then

```
6 dbms_output.put_line('Given number is Greater than Zero');
```

7 else

```
8 dbms_output.put_line('Given number is less than Zero');
```

```

9  end if;

```

```
10 end;
```



11 /

Enter value for n: -45

old 4: n:=&n;

new 4: n:=-45;

Given number is less than Zero

PL/SQL procedure successfully completed.

## **5. Nested IF-THEN-ELSE Statements**

SQL> declare

2 n number;

3 begin

4 n:=&n;

5 if n>0 then

6 dbms\_output.put\_line('Given number is Greater than Zero');

7 else

8 if n=0 then

9 dbms\_output.put\_line('Given number is Equal to Zero');

10 else

11 dbms\_output.put\_line('Given number is less than Zero');

12 end if;

13 end if;

14 end;

15 /

Enter value for n: 0

old 4: n:=&n;

new 4: n:=0;

Given number is Equal to Zero

PL/SQL procedure successfully completed.

## **6. IF-THEN-ELSIF Statement**

SQL> declare

2 n number;

```

3 begin
4 n:=&n;
5 if n>0 then
6 dbms_output.put_line('Given number is Greater than Zero');
7 elsif n=0 then
8 dbms_output.put_line('Given number is Equal to Zero');
9 else
10 dbms_output.put_line('Given number is less than Zero');
11 end if;
12 end;
13 /

```

Enter value for n: 56

old 4: n:=&n;

new 4: n:=56;

Given number is Greater than Zero

PL/SQL procedure successfully completed.

## **7. Extended IF-THEN Statement**

SQL> declare

```

2 grade char(1);
3 begin
4 grade:='B';
5 if grade='A' then
6 dbms_output.put_line('Excellent');
7 elsif grade='B' then
8 dbms_output.put_line('Very Good');
9 elsif grade='C' then
10 dbms_output.put_line('Good');
11 elsif grade='D' then
12 dbms_output.put_line('Fair');
13 elsif grade='F' then

```

```
14 dbms_output.put_line('Poor');
15 else
16 dbms_output.put_line('No such grade');
17 end if;
18 end;
19 /
```

Very Good

PL/SQL procedure successfully completed.

### **8. Simple CASE Statement**

SQL> declare

```
2 grade char(1);
3 begin
4 grade:='B';
5 case grade
6 when 'A' then dbms_output.put_line('Excellent');
7 when 'B' then dbms_output.put_line('Very Good');
8 when 'C' then dbms_output.put_line('Good');
9 when 'D' then dbms_output.put_line('Fair');
10 when 'F' then dbms_output.put_line('Poor');
11 else dbms_output.put_line('No such grade');
12 end case;
13 end;
14 /
```

Very Good

PL/SQL procedure successfully completed.

### **9. Searched CASE Statement**

SQL> declare

```
2 grade char(1);
3 begin
4 grade:='B';
```

```
5 case
6 when grade='A' then dbms_output.put_line('Excellent');
7 when grade='B' then dbms_output.put_line('Very Good');
8 when grade='C' then dbms_output.put_line('Good');
9 when grade='D' then dbms_output.put_line('Fair');
10 when grade='F' then dbms_output.put_line('Poor');
11 else dbms_output.put_line('No such grade');
12 end case;
13 end;
14 /
```

Very Good

PL/SQL procedure successfully completed.

#### **10. EXCEPTION Instead of ELSE Clause in CASE Statement**

```
SQL> declare
```

```
2 grade char(1);
3 begin
4 grade:='A';
5 case
6 when grade='A' then dbms_output.put_line('Excellent');
7 when grade='B' then dbms_output.put_line('Very Good');
8 when grade='C' then dbms_output.put_line('Good');
9 when grade='D' then dbms_output.put_line('Fair');
10 when grade='F' then dbms_output.put_line('Poor');
11 end case;
12 exception
13 when case_not_found then
14 dbms_output.put_line('No such Grade');
15 end;
16 /
```

Excellent

PL/SQL procedure successfully completed.

### **11. Program for simple LOOP**

SQL> declare

```
2 total number:=0;
3 begin
4 loop
5 total:=total+1;
6 exit when total>=5;
7 end loop;
8 dbms_output.put_line('total:'||total);
9 end;
10 /
```

total:5

PL/SQL procedure successfully completed.

### **12. Program using WHILE LOOP:**

SQL> declare

```
2 i number:=0;
3 j number:=0;
4 begin
5 while i<=100
6 loop
7 j:=j+i;
8 i:=i+2;
9 end loop;
10 dbms_output.put_line('The Value of j is '||j);
11 end;
12 /
```

The Value of j is 2550

PL/SQL procedure successfully completed.

### **13. WHILE-LOOP Statement**

SQL> declare

2 A number;

3 I number:=1;

4 begin

5 A:=10;

6 while I<A loop

7 dbms\_output.put\_line('Value:'||I);

8 I:=I+1;

9 end loop;

10 end;

11 /

Value:1

Value:2

Value:3

Value:4

Value:5

Value:6

Value:7

Value:8

Value:9

PL/SQL procedure successfully completed.

### **14. FOR-LOOP Statement**

SQL> begin

2 for i in 1..3

3 loop

4 dbms\_output.put\_line(to\_char(i));

5 end loop;

6 end;

7 /

1

2

3

PL/SQL procedure successfully completed.

### **15. Reverse FOR-LOOP Statement**

SQL> begin

2 for i in reverse 1..3

3 loop

4 dbms\_output.put\_line(to\_char(i));

5 end loop;

6 end;

7 /

3

2

1

PL/SQL procedure successfully completed.

### **16. Simple GOTO Statement**

SQL> declare

2 p varchar2(30);

3 n pls\_integer:=37;

4 begin

5 for j in 2..round(sqrt(n))

6 loop

7 if n mod j=0 then

8 p:='is not a prime number';

9 goto print\_now;

10 end if;

11 end loop;

12 p:='is a prime number';

```
13 <<print_now>>
14 dbms_output.put_line(to_char(n) || p);
15 end;
16 /
```

37 is a prime number

PL/SQL procedure successfully completed.

### **17. GOTO Statement to Branch to an Enclosing Block**

SQL> declare

```
2  v_last_name varchar(15);
3  v_emp_id number(6) := 1;
4  begin
5  <<get_name>>
6  begin
7  select name into v_last_name from instructor where id = v_emp_id;
8  dbms_output.put_line(v_last_name);
9  v_emp_id := v_emp_id + 2;
10 if v_emp_id <= 15 then
11     goto get_name;
12 end if;
13 exception
14 when no_data_found then
15     dbms_output.put_line('No instructor found with ID ' || v_emp_id);
16 end;
17 end;
18 /
```

dinesh

karthi

babu

lalith

hema



abarna

madan

suji

PL/SQL procedure successfully completed.

### **18. Do...While Statement:**

SQL> declare

```
2 num number:=1;
```

```
3 begin
```

```
4 loop
```

```
5 dbms_output.put(num||',' );
```

```
6 num:=num+1;
```

```
7 exit when num>5;
```

```
8 end loop;
```

```
9 dbms_output.put_line('Final: '||num);
```

```
10 end;
```

```
11 /
```

1,2,3,4,5,Final: 6

PL/SQL procedure successfully completed.

### **19. Factorial value**

SQL> declare

```
2 n number:=5;
```

```
3 factorial number:=1;
```

```
4 begin
```

```
5 for i in 1..n loop
```

```
6 factorial:=factorial*i;
```

```
7 end loop;
```

```
8 dbms_output.put_line('Factorial of '|| n ||' is '||factorial);
```

```
9 end;
```

```
10 /
```

Factorial of 5 is 120

PL/SQL procedure successfully completed.

## **20. Prime Number Generation**

SQL> declare

```
2    v_limit number := 15;
3    isprime boolean;
4  begin
5    for i in 2..v_limit loop
6      isprime := true;
7      for j in 2..sqrt(i) loop
8        if mod(i, j) = 0 then
9          isprime := false;
10         exit;
11       end if;
12     end loop;
13
14     if isprime then
15       dbms_output.put_line(i || ' is a prime number');
16     end if;
17   end loop;
18 end;
19 /
```

2 is a prime number

3 is a prime number

5 is a prime number

7 is a prime number

11 is a prime number

13 is a prime number

PL/SQL procedure successfully completed.

## **21. Fibonacci Series**

SQL> declare

```

2 limit number:=10;
3 a number:=0;
4 b number:=1;
5 c number;
6 begin
7 dbms_output.put_line('Fibonacci Series: ');
8 dbms_output.put_line(a);
9 dbms_output.put_line(b);
10 for i in 3..limit loop
11 c:=a+b;
12 dbms_output.put_line(c);
13 a:=b;
14 b:=c;
15 end loop;
16 end;
17 /

```

Fibonacci Series:

```

0
1
1
2
3
5
8
13
21
34

```

PL/SQL procedure successfully completed.

## 22. Checking Palindrome

SQL> declare

```

2 original varchar(20) := 'malayalam';
3 result varchar(100);
4 begin
5 for i in reverse 1..length(original) loop
6 result := result || substr(original, i, 1);
7 end loop;
8 if original = result then
9 dbms_output.put_line(original || ' is palindrome');
10 else
11 dbms_output.put_line(original || ' is not a palindrome');
12 end if;
13 end;
14 /

```

malayalam is palindrome

PL/SQL procedure successfully completed.

### **23. Swap two numbers**

SQL> declare

```

2 num1 number := 10;
3 num2 number := 20;
4 temp number;
5 begin
6 dbms_output.put_line('Before Swapping:');
7 dbms_output.put_line('Number 1: ' || num1);
8 dbms_output.put_line('Number 2: ' || num2);
9 temp := num1;
10 num1 := num2;
11 num2 := temp;
12 dbms_output.put_line('After Swapping:');
13 dbms_output.put_line('Number 1: ' || num1);
14 dbms_output.put_line('Number 2: ' || num2);

```

```
15 end;
```

```
16 /
```

Before Swapping:

Number 1: 10

Number 2: 20

After Swapping:

Number 1: 20

Number 2: 10

PL/SQL procedure successfully completed.

#### **24. PL/SQL block for updating single row into a table.**

```
SQL> select * from empdet;
```

ENO	ENAME	DEPTNO	BASIC	HRA	DA	PF	NETPAY
1	Dinesh	3	1000	500	200	70	1630

```
SQL> declare
```

```
2 v_eno empdet.eno%type;
```

```
3 v_ename empdet.ename%type;
```

```
4 v_deptno empdet.deptno%type;
```

```
5 v_basic empdet.basic%type;
```

```
6 v_hra empdet.HRA%type;
```

```
7 v_da empdet.DA%type;
```

```
8 v_pf empdet.PF%type;
```

```
9 v_netpay empdet.netpay%type;
```

```
10 begin
```

```
11 v_eno:=&v_eno;
```

```
12 v_ename:='&v_ename';
```

```
13 v_deptno:=&v_deptno;
```

```

14 v_basic:=&v_basic;
15 v_hra:=(v_basic*50)/100;
16 v_da:=(v_basic*20)/100;
17 v_pf:=(v_basic*7)/100;
18 v_netpay:=v_basic+v_hra+v_da-v_pf;
19 insert into empdet(eno,ename,deptno,basic,HRA,DA,PF,netpay)
values(v_eno,v_ename,v_deptno,v_basic,v_hra,v_da,v_pf,v_netpay);
20 dbms_output.put_line('Row inserted successfully.');
```

```

21 exception
22 when others then
23 dbms_output.put_line('Error:'|| SQLERRM);
24 end;
25 /
```

Enter value for v\_eno: 2

```
old 11: v_eno:=&v_eno;
```

```
new 11: v_eno:=2;
```

Enter value for v\_ename: Gunal

```
old 12: v_ename:='&v_ename';
```

```
new 12: v_ename:='Gunal';
```

Enter value for v\_deptno: 8

```
old 13: v_deptno:=&v_deptno;
```

```
new 13: v_deptno:=8;
```

Enter value for v\_basic: 25000

```
old 14: v_basic:=&v_basic;
```

```
new 14: v_basic:=25000;
```

Row inserted successfully.

PL/SQL procedure successfully completed.

```
SQL> select * from empdet;
```

ENO	ENAME	DEPTNO	BASIC	HRA	DA	PF	NETPAY
1	Dinesh	3	1000	500	200	70	1630

2 Gunal	8	25000	12500	5000	1750	40750
---------	---	-------	-------	------	------	-------

## 25. PL/SQL block for updating multiple rows into a table.

SQL> select \* from emp;

EID	NAME	SALARY	DNO
1	Dinesh	70000	3
2	Gunal	60000	4
3	Aanand	60000	2
4	Lalit	70000	1
5	Karthik	50000	3
6	Akhil	55000	4

6 rows selected.

SQL> declare

```
2 new_name varchar(15):='Dineshkumar';
3 begin
4 update emp set name=new_name where eid=1;
5 commit;
6 end;
7 /
```

PL/SQL procedure successfully completed.

SQL> select \* from emp;

EID	NAME	SALARY	DNO
1	Dineshkumar	70000	3
2	Gunal	60000	4
3	Aanand	60000	2
4	Lalit	70000	1
5	Karthik	50000	3
6	Akhil	55000	4

6 rows selected.

SQL> declare

2 new\_salary number:=80000;

3 begin

4 update emp set salary=new\_salary where salary>60000;

5 commit;

6 end;

7 /

PL/SQL procedure successfully completed.

SQL> select \* from emp;

EID	NAME	SALARY	DNO
1	Dineshkumar	80000	3
2	Gunal	60000	4
3	Aanand	60000	2
4	Lalit	80000	1
5	Karthik	50000	3
6	Akhil	55000	4

6 rows selected.

CONTENTS	MARKS ALLOTTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

### Result:

Thus, PL/SQL statements in various programs are executed successfully.



EX.NO:10

## CURSORS

30.05.2024

### AIM

To study and write the PL/SQL programs using cursors.

SQL> set serveroutput on

### 1. Program Using Implicit Cursor to Increase basic Of Each Customer By 500

SQL> select \* from employee;

ENAME	EID	DOB	BASIC	DA	TOTAL
-------	-----	-----	-------	----	-------

kavin	1	06-OCT-90	27500		
Arun	2	08-SEP-90	19500		
Sankar	3	01-JAN-89	14500		
Radha	4	10-APR-82	37500		
Kevin	5	02-FEB-89	32500		

SQL> declare

2 total\_rows number(2);

3 begin

4 update employee set basic=basic+500;

5 if sql%notfound then

6 dbms\_output.put\_line('no customers selected');

7 elsif sql%found then

8 total\_rows := sql%rowcount;

9 dbms\_output.put\_line( total\_rows || ' Employees selected ');

10 end if;

11 end;

12 /

5 Employees selected

PL/SQL procedure successfully completed.

SQL> select \* from employee;

ENAME	EID	DOB	BASIC	DA	TOTAL
kavin	1	06-OCT-90	28000		
Arun	2	08-SEP-90	20000		
Sankar	3	01-JAN-89	15000		
Radha	4	10-APR-82	38000		
Kevin	5	02-FEB-89	33000		

## 2. Program Using Explicit Cursor to Fetch the Employee Details from the Table

SQL> select \* from customers;

ID	NAME	ADDRESS	SALARY	AGE
1	Dinesh	Vriddhachalam	73500	24
2	Gunal	panruti	68500	24
3	Aanand	vellore	61500	30
4	Anu	chennai	70500	29
5	Allwin	salem	48500	27
6	vel	cuddalore	43500	26

6 rows selected.

SQL> declare

```
2  cursor c_customers is
3  select id, name, address from customers;
4  v_id customers.id%type;
5  v_name customers.name%type;
6  v_address customers.address%type;
7  begin
8  open c_customers;
9  loop
10  fetch c_customers into v_id, v_name, v_address;
```

```
11  exit when c_customers%notfound;
12  dbms_output.put_line(v_id || ' ' || v_name || ' ' || v_address);
13  end loop;
14  close c_customers;
15  end;
16 /
```

1 Dinesh Vriddhachalam

2 Gunal panruti

3 Aanand vellore

4 Anu chennai

5 Allwin salem

6 vel cuddalore

PL/SQL procedure successfully completed.

### **3. Program using explicit cursor to calculate da & total.**

**Set DA= 25% if salary <15000, DA=35% if salary <=15,000 and >30,000, DA=45% if  
Salary>=35,000**

SQL> declare

```
2  v_ename employee.ename%type;
3  v_eid employee.eid%type;
4  v_dob employee.dob%type;
5  v_basic employee.basic%type;
6  v_da employee.da%type;
7  v_total employee.total%type;
8  cursor c_employee is
9      select ename, eid, dob, basic
10     from employee;
11  begin
12      for emp_rec in c_employee loop
13          v_ename := emp_rec.ename;
14          v_eid := emp_rec.eid;
```

```

15     v_dob := emp_rec.dob;
16     v_basic := emp_rec.basic;
17
18     if v_basic < 15000 then
19         v_da := 0.25 * v_basic;
20     elsif v_basic <= 30000 then
21         v_da := 0.35 * v_basic;
22     else
23         v_da := 0.45 * v_basic;
24     end if;
25
26     v_total := v_basic + v_da;
27
28     dbms_output.put_line('employee name: ' || v_ename);
29     dbms_output.put_line('employee id: ' || v_eid);
30     dbms_output.put_line('dob: ' || to_char(v_dob, 'dd-mon-yyyy'));
31     dbms_output.put_line('basic salary: ' || v_basic);
32     dbms_output.put_line('da: ' || v_da);
33     dbms_output.put_line('total salary: ' || v_total);
34     dbms_output.put_line('-----');
35 end loop;
36 end;
37 /

```

employee name: kavin

employee id: 1

dob: 06-oct-1990

basic salary: 28000

da: 9800

total salary: 37800

-----

employee name: Arun

employee id: 2

dob: 08-sep-1990

basic salary: 20000

da: 7000

total salary: 27000

-----

employee name: Sankar

employee id: 3

dob: 01-jan-1989

basic salary: 15000

da: 5250

total salary: 20250

-----

employee name: Radha

employee id: 4

dob: 10-apr-1982

basic salary: 38000

da: 17100

total salary: 55100

-----

employee name: Kevin

employee id: 5

dob: 02-feb-1989

basic salary: 33000

da: 14850

total salary: 47850

-----

PL/SQL procedure successfully completed.

#### 4. Create an explicit cursor to calculate results of students

SQL> select \* from student;

NAME	ROLLNO	M1	M2	M3	TOTAL	AVERAGE RESULT
------	--------	----	----	----	-------	----------------

Dinesh	1	100	100	100		
Gunul	2	100	39	89		
Aanand	3	60	100	30		
Lalit	4	69	78	93		
Allwin	5	49	15	38		

SQL> declare

```
2   v_name student.name%type;
3   v_rno student.rollno%type;
4   v_m1 student.m1%type;
5   v_m2 student.m2%type;
6   v_m3 student.m3%type;
7   v_tot student.total%type;
8   v_avg student.average%type;
9   v_result student.result%type;
10
11  cursor c_students is
12      select name, rollno, m1, m2, m3
13      from student;
14  begin
15      for stud_rec in c_students loop
16          v_name := stud_rec.name;
17          v_rno := stud_rec.rollno;
18          v_m1 := stud_rec.m1;
19          v_m2 := stud_rec.m2;
20          v_m3 := stud_rec.m3;
```

```

21
22     v_tot := v_m1 + v_m2 + v_m3;
23     v_avg := v_tot / 3;
24
25     if v_avg >= 40 then
26         v_result := 'pass';
27     else
28         v_result := 'fail';
29     end if;
30
31     dbms_output.put_line('student name: ' || v_name);
32     dbms_output.put_line('roll number: ' || v_rno);
33     dbms_output.put_line('marks 1: ' || v_m1);
34     dbms_output.put_line('marks 2: ' || v_m2);
35     dbms_output.put_line('marks 3: ' || v_m3);
36     dbms_output.put_line('total marks: ' || v_tot);
37     dbms_output.put_line('average marks: ' || v_avg);
38     dbms_output.put_line('result: ' || v_result);
39     dbms_output.put_line('-----');
40 end loop;
41 end;
42 /

```

student name: Dinesh

roll number: 1

marks 1: 100

marks 2: 100

marks 3: 100

total marks: 300

average marks: 100

result: pass

-----  
student name: Gunal

roll number: 2

marks 1: 100

marks 2: 39

marks 3: 89

total marks: 228

average marks: 76

result: pass

-----  
student name: Aanand

roll number: 3

marks 1: 60

marks 2: 100

marks 3: 30

total marks: 190

average marks: 63.33

result: pass

-----  
student name: Lalit

roll number: 4

marks 1: 69

marks 2: 78

marks 3: 93

total marks: 240

average marks: 80

result: pass

-----  
student name: Allwin

roll number: 5



marks 1: 49

marks 2: 15

marks 3: 38

total marks: 102

average marks: 34

result: fail

-----

PL/SQL procedure successfully completed.

CONTENTS	MARKS ALLOTTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

## RESULT

Thus, PL/SQL programs using cursors are studied and executed.

EX.NO : 11

## TRIGGERS

06.06.2024

### AIM:

To create and access the database using triggers.

SQL> set serveroutput on;

### TO CREATE A TABLE REVISED

SQL> create table revised(empid number(10),name varchar(10),salary number(10));

Table created.

SQL> select \* from revised;

EMPID	NAME	SALARY
12	Gobi	25000
1	Dinesh	30000
2	Gunat	30000
3	Aanan	30000
4	Lalit	25000
5	Giri	25000
6	Allwin	5000
7	Tamil	30000
8	Karthi	30000
9	Suji	30000

10 rows selected.

### TO CREATE TRIGGER AND UPDATE THE SALARY VALUE

SQL> create or replace trigger t1

2 after insert on revised

3 begin

4 update revised set salary=40000 where salary>30000;

5 end;

6 /

Trigger created.

SQL> insert into revised values(10,'anu',35000);

1 row created.

### **TO DISPLAY THE TABLE REVISED AFTER UPDATING**

SQL> select \* from revised;

EMPID	NAME	SALARY
-----		
12	Gobi	25000
10	anu	40000
1	Dinesh	30000
2	Gunal	30000
3	Aanan	30000
4	Lalit	25000
5	Giri	25000
6	Allwin	5000
7	Tamil	30000
8	Karthi	30000
9	Suji	30000

11 rows selected.

### **TO DROP THE TRIGGER CREATED**

SQL> drop trigger t1;

Trigger dropped.

### **TO CREATE A TABLE NEWBACKUP**

SQL> create table newbackup(empid number(10),name varchar(10),salary number(10));

Table created.

### **TO CREATE A TABLE OLDBACKUP**

SQL> create table oldbackup(empid number(10),name varchar(10),salary number(10));

Table created.

## TO CREATE ANOTHER TRIGGER

SQL> create or replace trigger t2

2 after insert or update on revised

3 for each row

4 begin if inserting

5 then insert into newbackup values(:new.empid,:new.name,:new.salary);

6 elsif updating then insert into oldbackup values(:old.empid,:old.name,:old.salary);

7 end if;

8 end;

9 /

Trigger created.

## TO INSERT VALUES INTO THE TABLE REVISED

SQL> insert into revised values(11,'arun',35000);

1 row created.

## TO UPDATE THE FIELD SALARY IN THE TABLE REVISED

SQL> update revised set salary=35000 where empid=2;

1 row updated.

SQL> select \* from revised;

EMPID	NAME	SALARY
12	Gobi	25000
10	anu	40000
11	arun	35000
1	Dinesh	30000
2	Gunat	35000
3	Aanan	30000
4	Lalit	25000
5	Giri	25000
6	Allwin	5000

7 Tamil	30000
8 Karthi	30000
9 Suji	30000

12 rows selected.

### TO DISPLAY THE TABLE OLDBACKUP

SQL> select \* from oldbackup;

EMPID NAME	SALARY
-----	
2 Gunal	30000

### TO DISPLAY THE TABLE NEWBACKUP

SQL> select \* from newbackup;

EMPID NAME	SALARY
-----	
11 arun	35000

### 1. Create Statement level before insert trigger

SQL> create or replace trigger statment\_level\_trigger

2 before insert on faculty

3 for each row

4 declare

5 min\_salary constant number:=5000;

6 begin

7 if :new.salary<min\_salary then

8 raise\_application\_error(-20001,'Salary cannot be less than ' || min\_salary);

9 end if;

10 end;

11 /

Trigger created.

SQL> select \* from faculty;

FID	FNAME	DEPT	SALARY
1	Dinesh	IT	80000
2	Gunal	CSE	70000
3	Aanand	ECE	65000
4	Karthik	CSD	60000
5	Akhil	AIDS	55000
6	Allwin	IT	75000
7	Siva	MECH	50000
8	lalit	IT	5500

8 rows selected.

SQL> insert into faculty values(9,'anu','AI',4000);

insert into faculty values(9,'anu','AI',4000)

\*

ERROR at line 1:

ORA-20001: Salary cannot be less than 5000

ORA-06512: at "DINESHKUMAR.STATMENT\_LEVEL\_TRIGGER", line 5

ORA-04088: error during execution of trigger

'DINESHKUMAR.STATMENT\_LEVEL\_TRIGGER'

SQL> desc faculty\_log;

Name	Null?	Type
-----		
FID		NUMBER
FNAME		VARCHAR2(50)
DEPT		VARCHAR2(50)
OLD_SALARY		NUMBER
NEW_SALARY		NUMBER
UPDATE_TIME		DATE

**2. Create a Row level after update trigger to insert the new values into another table also**

SQL> create or replace trigger trg\_after\_update\_faculty

2 after update on faculty

3 for each row

4 begin

5 insert into faculty\_log(fid,fname,dept,old\_salary,new\_salary,update\_time)

6 values(:old.fid,:old.fname,:old.dept,:old.salary,:new.salary,sysdate);

7 end;

8 /

Trigger created.

SQL> update faculty set salary=90000 where fid=1;

1 row updated.

SQL> select \* from faculty;

FID	FNAME	DEPT	SALARY
-----			
1	Dinesh	IT	90000
2	Gunul	CSE	70000

3 Aanand	ECE	65000
4 Karthik	CSD	60000
5 Akhil	AIDS	55000
6 Allwin	IT	75000
7 Siva	MECH	50000
8 lalit	IT	5500

8 rows selected.

SQL> select \* from faculty\_log;

FID	FNAME	DEPT	OLD_SALARY	NEW_SALARY	UPDATE_TI
1	Dinesh	IT	80000	90000	01-JUN-24

CONTENTS	MARKS ALLOTTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

## RESULT:

Thus various types of trigger are created for the database.



EX.NO :12

## PROCEDURES AND FUNCTIONS

06.06.2024

### AIM:

To implement the procedure and functions using PL/SQL

### SQL FUNCTIONS AND PROCEDURES:

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms:

- **Functions:** these subprograms return a single value, mainly used to compute and return a value.
- **Procedures:** these subprograms do not return a value directly, mainly used to perform an action.

SQL> set serveroutput on;

#### 1. Create a function to find factorial of a number.

SQL> create or replace function factorial(n number) return number is

```
2  fact number := 1;
3  begin
4  if n < 0 then
5  return null;
6  elsif n = 0 then
7  return 1;
8  else
9  for i in 1..n loop
10     fact := fact * i;
11  end loop;
12  return fact;
13  end if;
14 end;
15 /
```

Function created.

```
SQL> select factorial(5) from dual;
```

```
FACTORIAL(5)
```

```
-----
```

```
120
```

## **2. Create a procedure to calculate the area of the square**

```
SQL> create or replace function area_of_square(a number) return number is
```

```
2 area number:=0;
```

```
3 begin
```

```
4 area:=a*a;
```

```
5 return area;
```

```
6 end;
```

```
7 /
```

Function created.

```
SQL> select area_of_square(5) from dual;
```

```
AREA_OF_SQUARE(5)
```

```
-----
```

```
25
```

## **3. Write a PL/SQL function called POW that takes two numbers as argument and return the value of the first number raised to the power of the second.**

```
SQL> create or replace function pow(base number,exponent number) return number is
```

```
2 result number:=1;
```

```
3 begin
```

```
4   for i in 1..exponent loop
```

```
5       result:=result*base;
```

```
6   end loop;
```

```
7   return result;
```

```
8 end;
```

```
9 /
```

Function created.

```
SQL> select pow(2,8) from dual;
```

POW(2,8)

-----

256

**4. Write a PL/SQL function ODDEVEN to return value TRUE if the number passed to it is EVEN else will return FALSE**

```
SQL> create or replace function oddeven(n number) return varchar is
```

```
2 begin
3   if mod(n,2)=0 then
4     return 'even';
5   else
6     return 'odd';
7   end if;
8 end;
9 /
```

Function created.

```
SQL> select oddeven(8) from dual;
```

ODDEVEN(8)

-----

even

```
SQL> select oddeven(11) from dual;
```

ODDEVEN(11)

-----

Odd

**5. Write a PL/SQL procedure called MULTI\_TABLE that takes two numbers as parameter and displays the multiplication of the first parameter till the second parameter.**

SQL> create or replace procedure multi\_table(n in number, m in number) is

```
2  result number;
3  begin
4  for i in n..m loop
5      result := n * i;
6      dbms_output.put_line(n || ' * ' || i || ' = ' || result);
7  end loop;
8  end;
9  /
```

Procedure created.

SQL> begin

```
2  multi_table(1,5);
3  end;
4  /
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
```

PL/SQL procedure successfully completed.

**6. Create a function to find the maximum salary from the table customer.**

SQL> select \* from customers;

ID	NAME	ADDRESS	SALARY	AGE
1	Dinesh	Vriddhachalam	70000	24

2 Gunal	panruti	65000	24
3 Aanand	vellore	58000	30
4 Anu	chennai	67000	29
5 Allwin	salem	45000	27
6 vel	cuddalore	40000	26

6 rows selected.

SQL> create or replace function maxsalary return number is

```

2  maximum_salary number;
3  begin
4  select max(salary) into maximum_salary from customers;
5  return maximum_salary;
6  end;
7  /

```

Function created.

SQL> select maxsalary() as maximum\_salary from dual;

MAXIMUM\_SALARY

-----

70000

## 7. Create a procedure to find the maximum and minimum salary from the table customer.

SQL> create or replace procedure max\_and\_min\_salary is

```

2  max_salary number;
3  min_salary number;
4  begin
5  select max(salary), min(salary) into max_salary, min_salary from customers;
6  dbms_output.put_line('Maximum Salary: ' || max_salary);
7  dbms_output.put_line('Minimum Salary: ' || min_salary);
8  end;
9  /

```

Procedure created.

SQL> begin

2 max\_and\_min\_salary;

3 end;

4 /

Maximum Salary: 70000

Minimum Salary: 40000

PL/SQL procedure successfully completed.

**8. Program for procedure – selected record's price is incremented by 500 , executing the procedure created and displaying the updated table**

SQL> select \* from customers;

ID	NAME	ADDRESS	SALARY	AGE
-----				
1	Dinesh	Vriddhachalam	70000	24
2	Gunal	panruti	65000	24
3	Aanand	vellore	58000	30
4	Anu	chennai	67000	29
5	Allwin	salem	45000	27
6	vel	cuddalore	40000	26

6 rows selected.

SQL> create or replace procedure increment\_price as

2 begin

3 update customers

4 set salary=salary+500

5 end;

6 /

Warning: Procedure created with compilation errors.

SQL> create or replace procedure increment\_price as

```
2 begin
3   update customers set salary=salary+500;
4 end;
5 /
```

Procedure created.

SQL> execute increment\_price;

PL/SQL procedure successfully completed.

**9.Create a function to return the sum of the salary in the customer table whose age is greater then 25.**

SQL> select \* from customers;

ID	NAME	ADDRESS	SALARY	AGE
1	Dinesh	Vriddhachalam	70500	24
2	Gunal	panruti	65500	24
3	Aanand	vellore	58500	30
4	Anu	chennai	67500	29
5	Allwin	salem	45500	27
6	vel	cuddalore	40500	26

6 rows selected.

SQL> create or replace function sumofsalary return number is

```
2   total_salary number:=0;
3   begin
4     select sum(salary) into total_salary from customers
5     where age>25;
6
7     return total_salary;
8 end;
9 /
```

Function created.

```
SQL> select sumofsalary as total_salary from dual;
```

```
TOTAL_SALARY
```

```
-----
```

```
212000
```

**10. Program for function– selected record's price is incremented by 1000 , executing the procedure created and displaying the updated table**

```
SQL> select * from customers;
```

ID	NAME	ADDRESS	SALARY	AGE
1	Dinesh	Vriddhachalam	72500	24
2	Gunal	panruti	67500	24
3	Aanand	vellore	60500	30
4	Anu	chennai	69500	29
5	Allwin	salem	47500	27
6	vel	cuddalore	42500	26

6 rows selected.

```
SQL> create or replace function increment_salary(p_salary number) return number is
```

```
2 begin
```

```
3   return p_salary + 1000;
```

```
4 end;
```

```
5 /
```

Function created.

```
SQL> begin
```

```
2   for rec in (select id, salary from customers) loop
```

```
3     update customers
```

```
4     set salary = increment_salary(rec.salary)
```

```
5     where id = rec.id;
```

```
6   end loop;
```



7 commit;

8 end;

9 /

PL/SQL procedure successfully completed.

SQL> select \* from customers;

ID	NAME	ADDRESS	SALARY	AGE
1	Dinesh	Vriddhachalam	73500	24
2	Gunal	panruti	68500	24
3	Aanand	vellore	61500	30
4	Anu	chennai	70500	29
5	Allwin	salem	48500	27
6	vel	cuddalore	43500	26

6 rows selected.

CONTENTS	MARKS ALLOTTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

### RESULT:

Thus the procedure and functions are executed successfully.

13.06.2024

**AIM:**

To develop a mini project an employee data management system that allows adding, editing, and deleting employee records with their details like name, role, address, and phone number.

**PROCEDURE:**

Front-end (HTML, CSS, JavaScript):

1. Create an HTML file with a table to display employee data and a form for adding/editing employees.
2. Use HTML tables or divs to structure the employee list and form.
3. Add input fields (text boxes, dropdowns) in the form for employee details.
4. Style the UI with CSS for better appearance and usability.
5. Use JavaScript for form validation and making AJAX requests.
6. Implement functions in JavaScript to update the employee list table based on server responses.

Back-end (MySQL, Server-side Language like PHP or Node.js):

1. Set up a MySQL database and create a table for employee data.
2. Install and configure the server-side language (e.g., PHP, Node.js).
3. Establish a database connection from the server-side code.
4. Develop server-side functions or APIs for CRUD operations:
  - Create (add new employee)
  - Read (retrieve employee data)
  - Update (edit employee information)
  - Delete (remove employee record)
5. Implement input validation and sanitization on the server-side.
6. Write SQL queries to interact with the database based on CRUD operations.
7. Handle AJAX requests from the front-end and send responses (e.g., JSON data) after executing SQL queries.

Integration and Testing:

1. Integrate the front-end and back-end components.
2. Test the application, including form validations, CRUD operations, and edge cases.
3. Implement error handling and display appropriate messages to the user.
4. Test for potential vulnerabilities like SQL injection.
5. Optimize the application's performance.

Deployment:

1. Deploy the front-end files (HTML, CSS, JavaScript) on a web server.
2. Deploy the back-end code and configure the server to handle requests.
3. Set up the MySQL database on a server or use a cloud-based service.
4. Configure the back-end to connect to the database.
5. Test the deployed application thoroughly.

## **CODING:**

### **customer.ejs**

```
<%- include("partials/_header") %>

<div class="container man mt-3">
  <div class="row">
    <div class="col-md-8">
      <table class="table">
        <thead>
          <tr>
            <th style="background-color: #000;color:#fff">Sno</th>
            <th style="background-color: #000;color:#fff">CId</th>
            <th style="background-color: #000;color:#fff">Name</th>
            <th style="background-color: #000;color:#fff">Role</th>
            <th style="background-color: #000;color:#fff">Address</th>
            <th style="background-color: #000;color:#fff">Phone</th>
            <th style="background-color: #000;color:#fff">Actions</th>
          </tr>
        </thead>
        <tbody>
          <% if (customers) { %> <% for(var i = 0; i < customers.length; i++) { %>
            <tr>
              <td><%= i+1 %></td>
              <td><%= customers[i].cid %></td>
              <td><%= customers[i].name %></td>
              <td><%= customers[i].role %></td>
              <td><%= customers[i].address %></td>
              <td><%= customers[i].phone %></td>
              <td class="d-flex gap-2">
                <a href="/update/<%= customers[i].id %>" class="btn btn-info">
                  Edit
                </a>
                <a href="/delete/<%= customers[i].id %>" class="btn btn-danger">
                  Delete
                </a>
              </td>
            </tr>
          <% } %> <% } %>
        </tbody>
      </table>
```

```
</div>
<div class="col-md-3 mn">
  <div class="card-body">
    <form action="/add" method="POST">
      <input
        type="number"
        name="cid"
        placeholder="Id"
        class="form-control mb-2"
        autofocus
      />
      <input
        type="text"
        name="name"
        placeholder="Name"
        class="form-control mb-2"
      />
      <input
        type="text"
        name="role"
        placeholder="Role"
        class="form-control mb-2"
      />
      <input
        type="text"
        name="address"
        placeholder="Address"
        class="form-control mb-2"
      />
      <input
        type="text"
        name="phone"
        placeholder="Phone"
        class="form-control mb-2"
      />

      <button type="submit" class="btn btn-info">save employee</button>
    </form>
  </div>
</div>
</div>
</div>
<%- include("partials/_footer") %>
```

```

customerController.js
import { pool } from "../db.js";

export const renderCustomers = async (req, res) => {
  const [rows] = await pool.query("SELECT * FROM customer");
  res.render("customers", { customers: rows });
};

export const createCustomers = async (req, res) => {
  const newCustomer = req.body;
  await pool.query("INSERT INTO customer set ?", [newCustomer]);
  res.redirect("/");
};

export const editCustomer = async (req, res) => {
  const { id } = req.params;
  const [result] = await pool.query("SELECT * FROM customer WHERE id = ?", [
    id,
  ]);
  res.render("customers_edit", { customer: result[0] });
};

export const updateCustomer = async (req, res) => {
  const { id } = req.params;
  const newCustomer = req.body;
  await pool.query("UPDATE customer set ? WHERE id = ?", [newCustomer, id]);
  res.redirect("/");
};

export const deleteCustomer = async (req, res) => {
  const { id } = req.params;
  const result = await pool.query("DELETE FROM customer WHERE id = ?", [id]);
  if (result.affectedRows === 1) {
    res.json({ message: "Customer deleted" });
  }
  res.redirect("/");
};

```

## OUTPUT:

EMPLOYEE DETAILS

SNO	CID	NAME	ROLE	ADDRESS	PHONE	ACTIONS
1	1	Giri	HR	ERODE	9034567812	<button>EDIT</button> <button>DELETE</button>
2	2	DINESHKUMAR V	Manager	VDM	8870099495	<button>EDIT</button> <button>DELETE</button>

SAVE EMPLOYEE

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- atm
- bank
- bill\_calculator
- bookmytickets
- book
- contact
- customersdb
  - Tables
    - customer
    - customer
    - Views
    - Stored Procedures
    - Functions
- database1
- database2
- dbms
- movie\_ticket\_booking\_system
- parking
- store
- sys
- timetable\_db

Administration Schemas

Information

No object selected

Object Info Session

SQL File 37

1 • select \* from customer;

Limit to 1000 rows

Result Grid

id	cid	name	role	address	phone
23	1	Giri	HR	ERODE	9034567812
25	2	DINESHKUMAR V	Manager	VDM	8870099495

customer 1 x

Apply Revert Context Help Shippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	22:49:22	select * from customer LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

CONTENTS	MARKS ALLOTED	MARKS OBTAINED
Aim, Algorithm, SQL, PL/SQL	30	
Execution and Result	20	
Viva	10	
Total	60	

### **RESULT:**

Thus, the mini project on the Employee Data Management System was developed successfully by using html,javascript,and MYSQL.

